ROBOTIC GRASPING OF ORBITAL REPLACEMENT UNITS

by

Helen Greiner

Submitted to the Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING

at the Massachusetts Institute of Technology

June 1989

Signature Redacted

Author______________________________
Department of Mechanical Engineering
May 5 1989

Signature Redacted

Certified by______________________________
Professor K. Yousef-Toumi
Associate Professor of Mechanical Engineering
Thesis Supervisor

Signature Redacted

Accepted by______________________________
Professor Peter Griffith
Chairman, Department Committee

ROBOTIC GRASPING OF ORBITAL REPLACEMENT UNITS

by

Helen Greiner

Submitted to the
Department of Mechanical Engineering

May 5 1989

In Partial Fulfillment of the Requirements for the Degree of
BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING

Abstract

In the presence of uncertainties in the location of objects with respect to a robot end-effector, the grasping of objects becomes a non-trivial task. One method of eliminating these uncertainties is to use the force-torque data produced by the contact between the gripper and object to locate the object and to align the gripper fingers with it.

Using this control method, a strategy for grasping the handling fixture of Orbital Replacement Units (ORU's) was developed. This strategy simplifies the alignment problem by eliminating the errors in each degree of freedom separately. Portions of this strategy have been implemented within the Telerobot Testbed facility at JPL to collect experimental data. This data was then used to analyze and evaluate system performance and to identify control issues which still remain to be addressed.

Thesis Supervisor: Professor K. Yousef-Toumi
Title: Associate Professor of Mechanical Engineering

ACKNOWLEDGEMENTS

This work was performed at the Jet Propulsion Laboratory through the Engineering Internship Program. I would like to thank the Tele-Autonomous Systems Group for their support and help. I would like to especialy thank my supervisor Henry Stone for his time, expertise, and encouragement. The summer was an exceptional learning experience. Professor Yousef-Toumi was very helpful in editting and clarify this document. And, last but most important, I dedicate this thesis to my parents for their support and encouragement.

Contents

1 **Introduction** **8**
1.1 Background 8
1.2 Project Description 10
1.3 Thesis Outline 11

2 **Grasping** **13**

3 **Grapple Lug** **15**
3.1 Finger and Lug Mechanical Design 15
3.2 Finger/Lug Construction 16
3.3 Finger/Lug Misalignment 17

4 **Algorithms** **24**
4.1 Introduction 24
4.2 Translational Position Accommodation 25
4.3 Rotational Position Accommodation 26
4.4 Relieve Contact Forces 27
4.5 Direct Force Feedback 27

5 **Control Strategy** **29**
5.1 Introduction 29
5.2 Approach 30
5.3 Alignment 30

5.3.1 Point of Contact Identification 30

5.3.2 Surface Alignment 31

5.4 Inward Finger Motion 32

6 Implementation **35**

6.1 System Configuration 35

6.1.1 Introduction 35

6.1.2 LORD Instrumented Gripper 36

6.1.3 LORD Wrist Force-Torque Sensor 37

6.1.4 Telerobot TestBed Control Structure 37

6.2 Frames 37

6.3 Robot Control Language 38

6.4 Sensory Information 39

6.4.1 Introduction 39

6.4.2 Force Transformations 40

6.4.3 Sensor Biasing 41

6.4.4 Transient Dynamic Loading 41

6.5 Performance Evaluation 41

6.5.1 Approach Motion 42

6.5.2 Instantaneous Contact Forces 43

6.5.3 Point of Contact Identification 43

6.5.4 Levelling 45

7 Conclusions and Recommendations **52**

References **55**

Appendix **57**

List of Figures

1-1 Typical ORU 12

3-1 a. Standard Grapple Lug, b. Modified Grapple Lug 19
3-2 Specialized Fingers 20
3-3 Finger and Grapple Lug Design 20
3-4 Fingers Perfectly Aligned with Lug 21
3-5 Critical Position and Orientation Error Tolerances 22
3-6 Positioning Tolerance in Z Direction 23

5-1 Point of Contact 34

6-1 System Configuration 46
6-2 Finger and Grapple Lug Frames 47
6-3 Normal Force at a Velocity of 3.6 mm/sec without Filtering 47
6-4 Normal Force at a Velocity of 3.6 mm/sec with Filtering 48
6-5 Normal Force at a Velocity of 18 mm/sec without Filtering 48
6-6 Normal Force at a Velocity of 18 mm/sec 49
6-7 Normal Force at a Velocity of 3.6 mm/sec 49
6-8 Guarded Motion; Speed = 3.6 mm/s; Threshold = 50 FTU's) 50
6-9 Forward Motion Posision Accommodation; Speed = 3.6 mm/s; Threshold = 50 FTU's 50
6-10 Special Tool used to Implement Steps of Grasping Operation 51

7-1 Prototype Grapple Lug and Finger Design 54

List of Tables

3.1 Critical Position and Orientation Error Tolerances 18

6.1 Performance of Point of Contact Algorithm 44

Chapter 1

Introduction

1.1 Background

The Earth Observing System (EOS) is a scientific experimentation satellite which is currently in the design phase at the Jet Propulsion Laboratory. This satellite contains four platforms which hold scientific experiments. Each experimental setup is configured on a separate module called an Orbital Replacement Unit (ORU). Figure 1-1 shows the design of a typical unit. The parts of the ORU of interest to this study are the connector and the grappling fixture. The connector is a Standard Interface Connector (SIC) which consists of an active part which protrudes from beneath the ORU and a passive part on the EOS platform. The grappling fixture retains the physical similarity to the grasping fixtures currently in use, but was modified for a robotic grasp (see Chapter 3 for details). The EOS satellite also contains a stow-bin for storing additional ORU's which are used to replace the original ORU's when their associated experiments either terminate or malfunction. Hence, there is a need to be able to interchange the ORU's. In the current design, the replacement of an ORU can be broken down into three steps as follows:

1. Transporting ORU

2. Positioning the ORU with resect to the EOS platform

3. Fastening the ORU to the platform

First, the ORU's are picked up and moved to a predetermined location on the EOS platform. Second, the two pins of the SIC are inserted into the holes in the EOS platform. Thirdly, the SIC bolt located in the center of the grappling fixture is tightened. To remove an ORU, these steps are performed in the reverse order.

Currently in space applications, tasks like the ORU changeout are performed by astronauts. This technique, called EVA, means the astronauts must leave the spacecraft. However, the EOS satellite will be in polar orbit where the harsh radiation conditions make standard EVA unfeasible. Automated machinery could be used to perform the pick-and-place tasks if little uncertainty is present in the location of objects. Such a system would require the design of a simpler means of connecting the ORU's to the EOS platform, and would provide no recourse if an ORU became misoriented or wrongly positioned. In addition, it is desirable to incorporate flexibility into the EOS design enabling it to perform simple experiment servicing tasks such as wiping dirty optical lenses. Hence, the EOS design team is considering the use of robotic manipulators on the EOS satellite. At the request of the EOS design team, a study evaluating the use of available robotic technology to perform the EOS module servicing tasks is being performed in the Tele-Autonomous Testbed Facility at JPL. A brief description of the Tele-Autonomous Testbed Facility and the work being done for the EOS project is given to provide context for this report.

The Tele-Autonomous Systems (TAS) Group is one of several groups at JPL responsible for conducting advanced research and development in the tele-operation and autonomous control of robots. One project that the TAS group is currently involved in is the Telerobot Testbed Project, whose long term goal is to demonstrate the ability of a robotic system to autonomously grapple and repair satellites. The Testbed integrates existing technologies in areas such as vision detection and tracking, trajectory planners, path planners, and force control, complient control, tele-operation, and shared control. In this testbed facility, two Unimation six degree-of-freedom PUMA 560 robots perform various manipulation tasks while a third Puma 560 with a mounted stereo camera system serves as an eye stalk for object verification. During the summer of 1988, the TestBed facility was used, in part, to research the robotic issues of concern to the EOS Project. The issues were divided into four major groups which were: manipulator kinematic complexity, control, precision and accuracy, and planning. The work involved both analysis and laboratory verification. Reports in these four areas form the deliverables to the EOS design team. This report discusses the grasping of the ORU modules which formed one part of the controls research.

1.2 Project Description

In order to manipulate an object, a robot must be able first to grasp it reliably. In most robot systems, the grasp is performed by an actuated gripper attached to the last link of the robot. The task of bringing the gripper into contact and reliably grasping an object is non-trivial in the presence of uncertainty in the object's location.[1] The robot must locate the object before a grasp can be performed. Many different approaches to accomplish this are possible such as: improving the models of the object and robot arm, teaching specific locations, or using sensory information (visual, proximity, or force-torque data) to determine the location of the object. However, all these methods present difficulties. Making the models of the system more precise or teaching specific locations would decrease the robustness since the system would not be able to accommodate even slight errors in object positioning. Using the vision system would increase the complexity of the system substantially, and the accuracy would be a function of how well the camera resolves object posions. In addition, the practical problem of the camera view being blocked by the gripper and fingers means that a control scheme for the camera stalk must be developed. Although using proximity sensors seems like a viable approach, the proper hardware was not available to us for this project. Therefore, we propose to use force-torque information from a sensor mounted on the robot arm to assist in performing the grasp. In addition, we will make the grasp more reliable by a gripper finger design which maximizes tolerances to position and orientation errors. Once the gripper is aligned

[1] Location specifies both the position and orientation of a frame.

with the object to within these tolerances, sensory information will be used to control the robot when performing the remaining alignment. This involves processing the sensory information and the real-time digital control of the robot.

This study is only concerned with grasping for a specific application, that is, the grasping of a grappling fixture on on ORU modules. In this report, the first steps of a grasping operation are analyzed, designed, and implemented. The grasping procedure developed in this report is a stand alone macro which will evolve into a primitive to be used by a higher level control system. As discussed previously, this work forms part of a study being performed for the EOS satellite on the use of available robotic technology for servicing ORU modules.[1]

1.3 Thesis Outline

Chapter 2 of this report relates this work to previous work performed in force control and grasping. The mechanics of the grasping operation are discussed in Chapter 3. The algorithms used to control the robot end-effector during the grasping operation are presented in Chapter 4. Chapter 5 discusses how these algorithms were integrated into a complete stategy for grasping. Implementation issues such as the system hardware and software and experimental data showing the performance of a few significant parts of the grasping operation are descibed in Chapter 6. Finally, conclusions and recommendations for further studies are presented in Chapter 7.

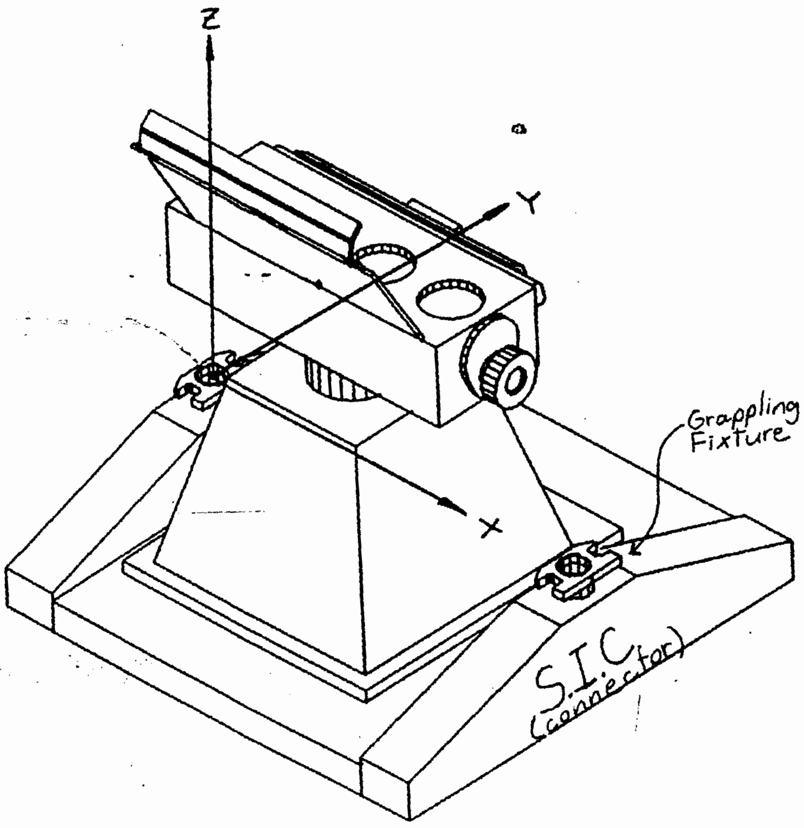

Figure 1-1: Typical ORU

Chapter 2

Grasping

Before the 1970s, research concerning the control of robotic manipulators centered on position control, that is control of a robotic end-effector as it moves through free space. However, in recent years, methods of force control have received attention because they increase the range of tasks that a robot can perform. Force control is usually employed when the arm is in contact with an object in its environment. Many different force control methods have been developed such as accommodation[2], active compliance[3], and hybrid force position control[4][5]. A general framework to look at force control has been established since the aforementioned control methods all integrate the same components: task description, a task execution strategy, command logic, and control and stability [6]. Each of these components will be examined individually with emphasis on how this project falls within the general framework.

The task description models the interactions between forces and motions during contact. In this project, the task is the grasping of an ORU. This is similar in many respects to the mating of parts during a mechanical assembly operation in the presence of position and orientation errors in the placement of the parts.

The task execution strategy uses the task description models to achieve desirable force, motion or position states. All the force control methods developed to date depend on humans providing the higher level reasoning. No automatic generation of strategies has been achieved [6]. Thus, for a specific application, like grasping ORU handling fixtures, a task execution strategy must be developed. This plan should be be formulated to insure reliable and robust task performance.

Command logic has been defined as "the choice of commanded forces or velocities or both in order to cause certain response forces or motions to arise"[6]. The choice of control parameters falls within this function. As part of this project, the performance of the task execution strategy was evaluated as a function of various control parameters such as feedback gains, threshold forces, and commanded velocities.

Force control of a robot means feedback of sensory information to modify forces or motions. Our sensory information was obtained from a LORD wrist force torque sensor. Use of a wrist sensor eliminates some of the problems involved in measuring joint motor currents or torques to obtain force readings. For example, any method of acquiring force data involving the robot joints, must compensate for friction and gravity using accurate manipulator models[4]. A wrist sensor decouples the effects of friction and gravity on the sensor readings from the robot's link motions. Consequently, only the dynamics of the sensor itself, the gripper, and the fingers must be considered in processing the sensory information.

The type of control used in this project was governed to a large extent by the robotic environment in the JPL Telerobot Testbed. In the Testbed, the real-time control software, RCCL[7], is used to plan trajectories and synchronize signals implementing real-time position control. The control logic can read the force information thus decision making and the choice of control parameters can be a function of the forces. In this control environment, we do not directly control the current in the robot joints. Therefore, the main focus of this project is using this environment to achieve a goal state (a reliable grasp) through planning. The particular design of the ORU handing fixture makes this a task which hasn't been considered before in the literature. It is hoped that the way in which the force-torque data is used is applicable to a variety of other tasks.

Chapter 3

Grapple Lug

To perform a grasp with a robotic manipulator, the standard ORU grapple lug design was modified as shown in Figure 3-1. Specialized robotic fingers were also machined to mate with this modified grapple lug (see Figure 3-2). A three dimensional sketch is provided in Figure 3-3 for visual clarity. The grapple lug is bolted to the EOS platform. The fingers are attached to the robotic gripper, and are interchangable by means of a spring-loaded fastening mechanism. In this chapter, we explain how the modification to the grapple lug make it compatible with a reliable robot grasp. Then, some machining details are discussed. Issues in the alignment of the finger and lug which are purely a function of this specific design are quantified in the last section of this chapter.

3.1 Finger and Lug Mechanical Design

The grapple lug which is currently used in space applications was modified to make it robot friendly. Maintaining physical resemblance to the currently used lug was important to the EOS design team. The modications to the grapple lug were influenced by the need to have rich force-torque information for implementing compliant control strategies. The design features of the fingers and modified lug are (as labelled in Figure 3-3):

- A - V-grooves on the edges of the grapple lug and corresponding wedges in the fingers

- B - a triangular piece in the fingers which fits into the cutout in the fingers
- C- flat surfaces on the bottom of the fingers and the top of the grapple lug

The V-grooves (A) are important for the following reasons. First, they maximize the amount of positioning errors the system can tolerate. Assuming the fingers are brought within the tolerances discussed in Section 3.3, a grasp is mechanically feasible. Also, if the positioning tolerances are not met (in this case in the Z direction), the position of the fingers after the grasp will allow the error to be detected. Second, the V-grooves align the fingers and the grapple lug in the Z direction as the fingers close. Thirdly, the sloped surfaces will generate useful control signals during the inward finger motion. The grasping algorithm combines position and force control. The control signals used in the force control (control when the fingers are in contact with the grapple lug) come from a six-axis force torque sensor. The V-grooves provide force-torque data rich enough to allow the robot to automatically comply in the Z direction to align the grooves.

The triangular section of the fingers (B) performs an analogous function in the Y direction. The only difference is that the contact between the fingers and the grapple lug is a line as opposed to a plane. Contact between the surfaces (A) prevent misalignment about the Z axis. Because the contact is in a line the amount of sliding friction as the fingers close will be reduced.

Likewise, the flat surfaces (C) can be used to eliminate errors in 2 degrees of freedom. The gripper approaches the lug from the positive Z direction. One control scheme is to first bring the flat surface of the fingers (C) into contact with the flat surface of the grapple lug and then to rotate about the point of contact to bring the to flat surfaces flush. This motion would eliminate orientation errors in the X-Y plane.

3.2 Finger/Lug Construction

The fingers were machined as four separate pieces labelled numerically in Figure 3-2. This reduced machining costs by eliminating the need to cut elaborate shapes and allowed the parts of the 3 sets of fingers to be machined in a batch. The four parts are fastened together by machine screws. Parts 1, 2, and 3 were constructed from aluminum. The primary motivation for using aluminium was to lower machining costs and to keep the payload on the Puma arm (including gripper, fingers, and ORU) under 5 lbs which is the rated payload for the Puma arms during motion. The one exception is part 4 - the flange which attaches to the spring loaded fastening mechanism. Steel was used for this part because it is only 3/16" thick, but it experiences the most torque during grasping.

3.3 Finger/Lug Misalignment

The ideal finger location prior to grasping is such that the grapple-lug is centered between the two fingers and rotationally aligned as shown in Figure 3-4. Given this initial positioning, by simply moving the fingers inward, they will mate perfectly with the grapple lug. However, because of position and orientation errors in the Testbed System, this ideal situation rarely occurs. If the position and/or orientation errors are sufficiently large, a grasp may not be physically realizable. It is thus important to understand the extent of location of errors which can be tolerated and still provide an opportunity for a successful grasp. This section discusses and quantifies the tolerances which must be met. If these tolerances are not met, the grasp will be guaranteed to fail. Because the force-torque data for a failed grasp may be indistinguishable from that of a correct grasp, the position of the fingers must be checked after the approach motion has been accomplished.

The tolerances in and about each axis are a function of the geometry of fingers shown in Figure 3-3. If these tolerances are met, compliant control strategies will be applied to eliminate the remaining position and orientation errors. The crucial tolerances are listed in Table 3.1. They were computed through geometrical analysis of the lug and fingers.

Cases I, II,V, and VI are fairly straightforward to visualize from Figure 3-5, which illustrates a top view of the fingers and grapple lug during these cases. These number are a function of the maximum finger opening. The maximum positioning error in the Z direction is 0.25 inches as shown in Figure 3-6. Figure 3-5 shows a side and a front view to illustrate the rotational error about the X and Y axis for cases IV and V, respectively.

In each of the cases position and orientation errors about each axis are examined

Cases	Description	Symbol	Value
I	Positional Error along X-axis	Δ X	0.37 in
II	Positional Error along Y-axis	Δ Y	0.34 in
III	Positional Error along Z-axis	Δ Z	0.25 in
IV	Rotational Error about X-axis	$\Delta\Theta_X$	11.1 deg
V	Rotational Error about Y-axis	$\Delta\Theta_Y$	6.3 deg
VI	Rotational Error about Z-axis	$\Delta\Theta_Z$	8.6 deg

Table 3.1: Critical Position and Orientation Error Tolerances

independently. However, a combination of two or more types of errors may reduce the tolerances considerably. The many variations possible make it difficult to analyze every combination. Therefore, we must take into account that the numbers listed in Table 3.1 are only upper-bounds for the tolerances in each degree-of-freedom.

Previous experience designing a calibration system for the Tele-robotic testbed indicated that the errors present in the Testbed System are less than necessary to meet these tolerances. The Testbed System experiences positioning errors of less than a few millimeters and less than a few degrees. Therefore, for the purpose of this project, the robot was not programmed recover from errors.

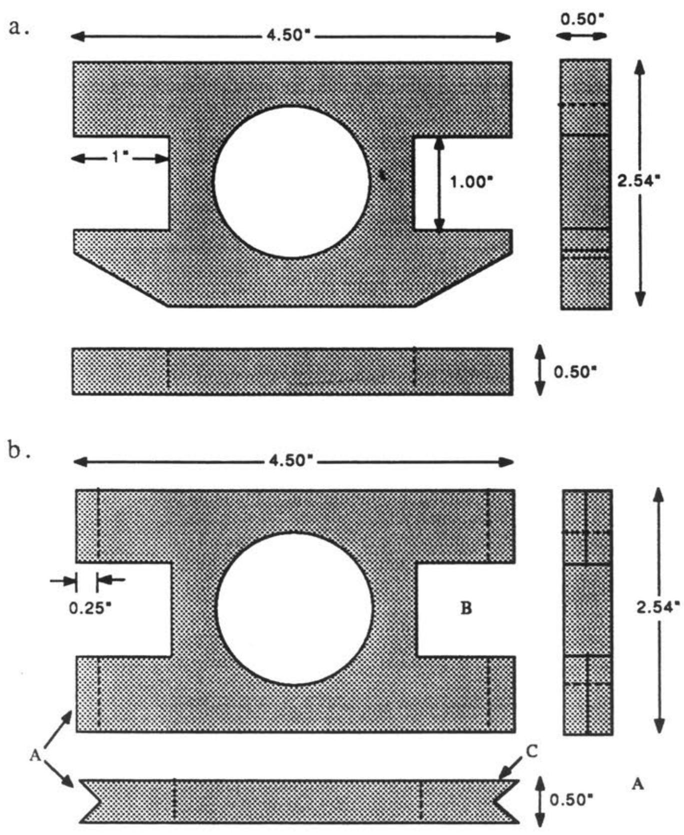

Figure 3-1: a. Standard Grapple Lug, b. Modified Grapple Lug

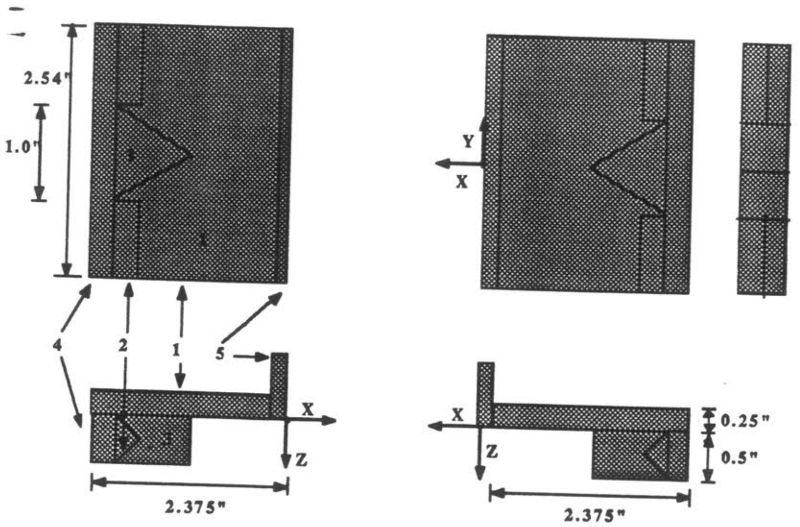

Figure 3-2: Specialized Fingers

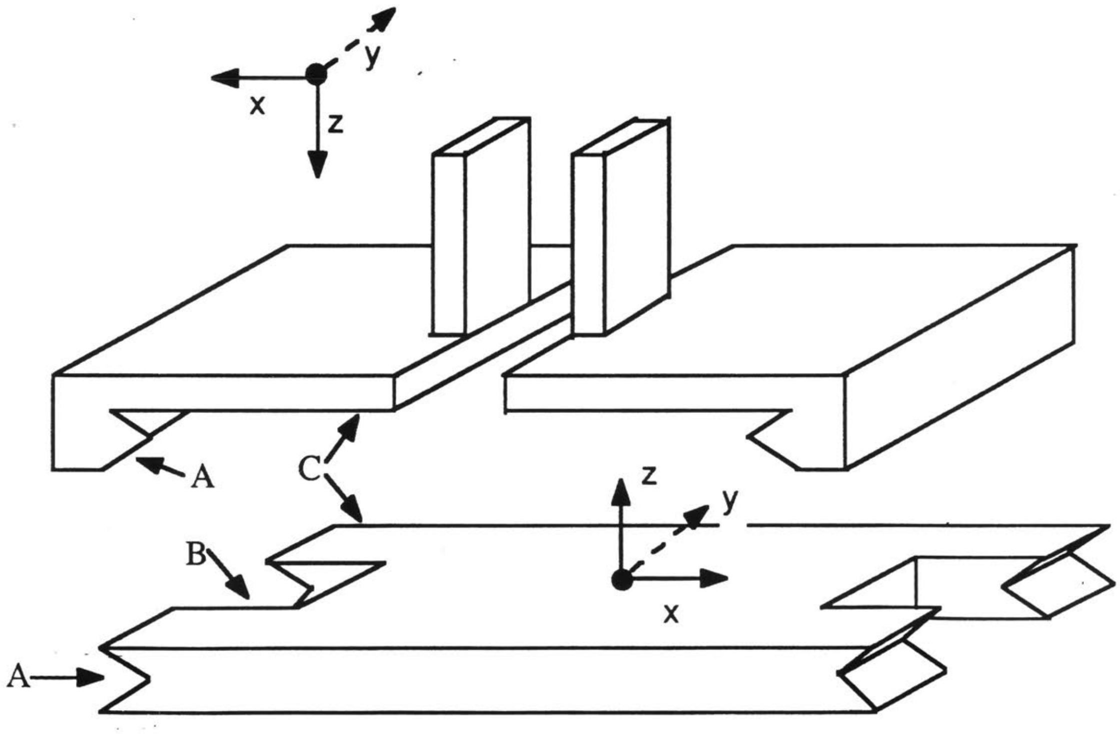

Figure 3-3: Finger and Grapple Lug Design

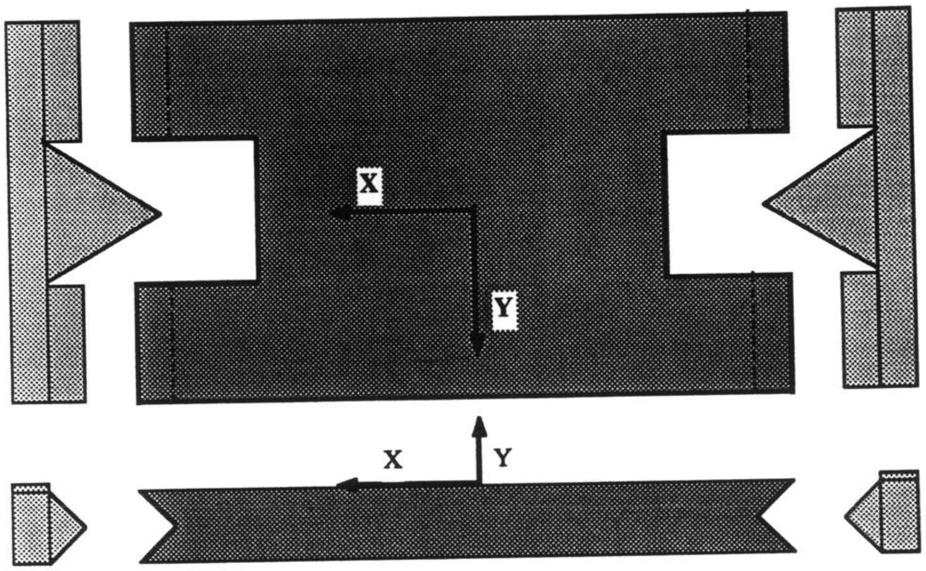

Figure 3-4: Fingers Perfectly Aligned with Lug

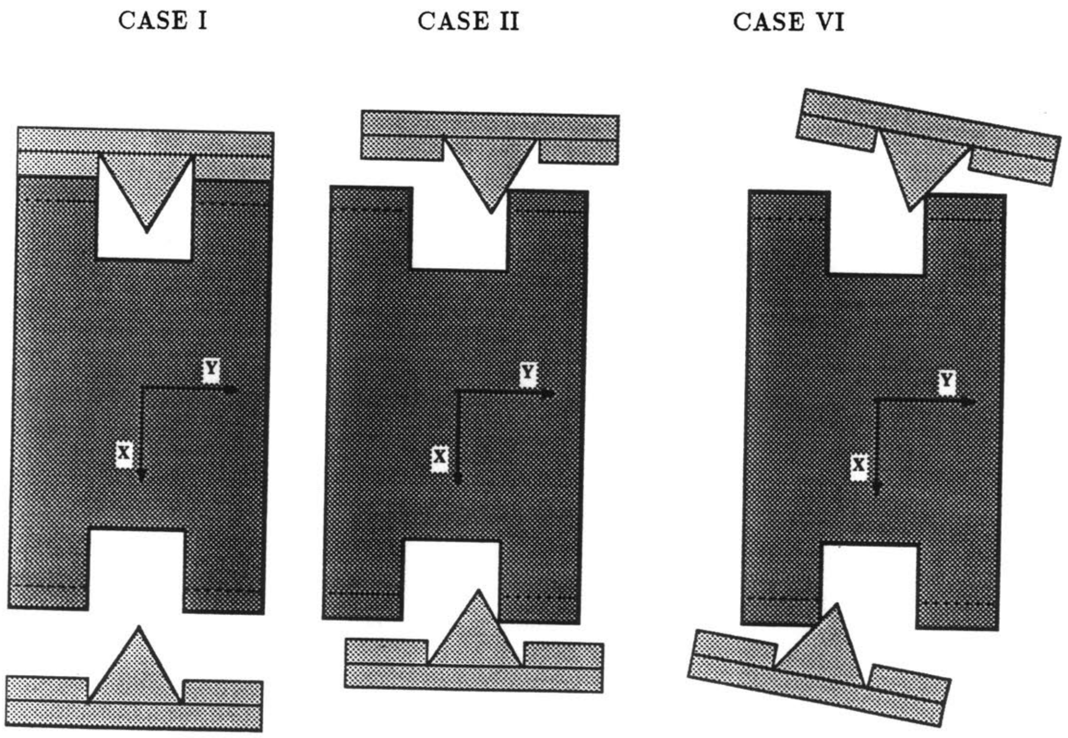

Figure 3-5: Critical Position and Orientation Error Tolerances

CASE III

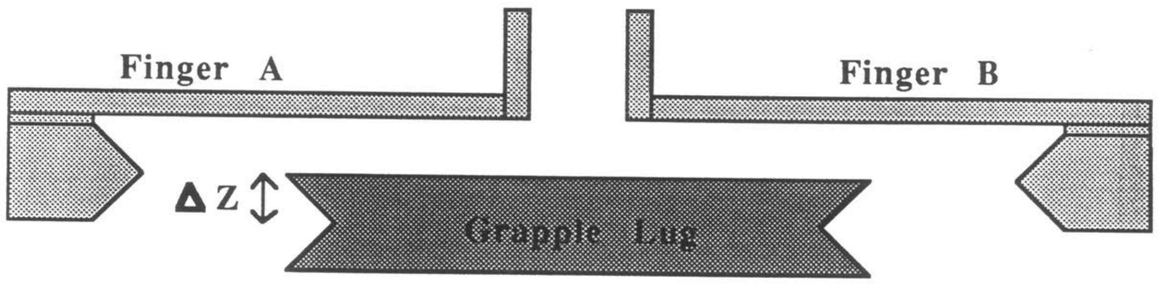

Figure 3-6: Positioning Tolerance in Z Direction

Chapter 4

Algorithms

4.1 Introduction

When the robot fingers come into contact with an object, some means of force control must be employed to avoid over-stressing either the robot joints or the force-torque sensor. System capabilities limit the type of force control available. In the current testbed facility, the hardware does not allow direct control of the current in the robot joints. Thus, we cannot use compliant control techniques which necessitate controlling the robot joint torques.

The type of force control used in performing the grasping operation is commonly referred to as position accommodation. In position accommodation, the feedback signals are the force and torque readings. Position accommodation consists of two parts: motion in free-space and motion during contact. A nominal driving increment moves the robot end-effector in free space. Upon contact with an object, control is accomplished by changing the driving increment by an amount proportional to the forces felt at the end-effector. These forces are obtained from a force-torque sensor mounted to the end-effector of the robot. The algorithms presented in this section assume that the forces sensed are entirely induced by contact. As discussed in Section 6.4 other forces may be present, however, we will assume that they can be eliminated or the effects are negligible.

This technique of force control can also be thought of as changing the position set-point that the robot will move to in the next sample period by an amount proportional to the forces, hence the name position accommodation. We will use three modes of position accommodation since our motion control strategy consists of three distinct types of motion: forward, rotational, and grasping. These control modes are Translational Position Accommodation, Rotational Position Accommodation, and Direct Force Feedback. Translation Position Accommodation moves the robot end-effector along an arbitrary vector at a speed proportional to the forces felt at the end effector. Rotational Position Accommodation moves the robot end-effector about an arbitrary vector at a speed proportional to the torques felt at the end-effector. In Direct Force Feedback Control, the motion is still proportional to the forces. However, in this mode, no driving increment is specified, thus motion is induced entirely by the force interaction.

4.2 Translational Position Accommodation

During translational position accommodation, the end-effector moves through free space at a constant velocity until contact occurs. Upon contact, the driving increment is decreased in proportion to the forces sensed at the end-effector. The control algorithm we use for the Translational Position Accommodation is

```
while absolute value of $\delta F < F_{thr}$
        $\delta F = (\vec{f_i} - \vec{f_{init}}) \cdot \vec{u}$
        $K = \frac{\Delta P}{F_{thr}}$
        $\vec{p_i} = \vec{p_{i-1}} + \Delta P \vec{u} - K \cdot \delta F \vec{u}$
        $\vec{r_i} = \vec{r_{i-1}}$
```

In the preceding algorithm, the vectors, $\vec{p_i}$ and $\vec{p_{i-1}}$, are vectors consisting of three components which describe the new position of the end-effector and the previous position, respectively. The rotational components of the location of the end-effector, denoted as $\vec{r_i}$, do not change during translational motion. The driving increment, ΔP, is a user set parameter describing the magnitude of the change in end-effector position the robot should make during each sample period. The unit vector, $\vec{u}$, is a three component vector describing the direction of motion in the end-effector coordinate frame. The difference between the initial forces due to gravity loading and the forces sensed during contact scaled along ~~the~~ direction of motion is δF. Prior to contact, δF is zero causing the arm to move in the direction $\vec{u}$ by an amount determined by the driving increment. Upon contact, δF increases proportionally to the forces sensed, and when δF is equal to the force threshold, F_{thr}, the end effector velocity is zero. If a force impulse occurs during contact, this routine will act like a guarded motion in the direction of motion because the threshold ,F_{thr}, will be achieved instantaneously.

4.3 Rotational Position Accommodation

Rotational Position Accommodation is analogous to the Translational Position Accommodation with the motion of the end-effector being a rotation about an arbitrary vector instead of motion along a vector. The motion termination threshold is now a torque, T_{thr}. A general implementation of this would stop on a threshold torque in the direction of motion similar to the Translational Position Accommodation. However, the task geometry makes it desirable to stop the rotation about the X and Y axis of the fingers independently. Hence, the following algorithm was implemented for Rotational Position Accommodation

```
while $t_x$ or $t_y < T_{thr}$
    if $t_x < T_{thr}$ then $u_x = 0$
    if $t_y < T_{thr}$ then $u_y = 0$
        $\delta T = (\vec{t_i} - \vec{t_{init}}) \cdot \vec{u}$
        $K = \frac{\Delta R}{T_{thr}}$
        $\vec{r_i} = \vec{r_{i-1}} + \Delta R \vec{u} - K \cdot \delta T \vec{u}$
        $\vec{p_i} = \vec{p_{i-1}}$
```

The 3 vector $\vec{r_i} = [r_{xi}, r_{yi}, r_{zi}]^T$ describes the differential rotation about each axis. The vector $\vec{r}$ is used to control the robot by concatenating $\Delta(\vec{r})$ to the position equation used to control the robot end-effector(see Section 6.3). Since we only expect errors of a few degrees, the following approximation was used to save computation time. This approximation discards high order terms, and consequently is valid only for small rotations

[8].

$$\Delta(\vec{r})_i = \begin{bmatrix} 0 & -r_{zi} & r_{yi} & p_{xi} \\ r_{zi} & 0 & -r_{xi} & p_{yi} \\ -r_{yi} & r_{xi} & 0 & p_{zi} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

4.4 Relieve Contact Forces

After the robot arm has achieved contact with an object in its environment, we sometimes wish to relieve the contact forces before the next motion is performed. There are two ways this can be done. One is to relieve the forces in all directions using the direct force feedback decribed in the next section. However, it may be desirable to reverse in the same direction that the arm was moving when contact occured. In this case, a simple backdriving routine to zero the forces is employed:

```
while $||\delta F|| > \epsilon$
        $\vec{p_{i+1}} = \vec{p_i} - K * \vec{u}$
```

As in the forward motion routine, the three component vector, $\vec{u}$, describes the direction of motion in the end-effector coordinate frame. The gain K specifies the amount move back during each sample period, and ϵ is a user specified tolerance about zero.

4.5 Direct Force Feedback

The position of the arm can also be made purely a function of the forces sensed:

$$\vec{p_i} = \vec{p_{i-1}} + K \cdot \vec{f}$$
$$\vec{r_i} = \vec{r_{i-1}} + K \cdot \vec{t}$$
$$\vec{f} = \vec{f_i} - \vec{f_{init}}$$
$$\vec{t} = \vec{t_i} - \vec{t_{init}}$$

The positions and rotations are independant for small angles. The gain, K, determines the reaction speed. A larger gain will make the response faster, and a negative gain would make the robot push with greater force against the contact. This algorithm will be employed while moving the gripper fingers inwards, thus relieving the forces induced as the fingers align with the grapple lug.

Chapter 5

Control Strategy

5.1 Introduction

A successful grasping operation is characterized by a sequence of motions which bring the gripper fingers into contact with the grapple lug and align the gripper fingers such that they mate correctly with the grooves in the grapple lug. This sequence can be divided into the following 5 steps:

1. the approach
2. alignment
3. inward finger motion
4. center
5. ungrasp and depart

This work assumes that a higher level control system will bring the gripper into a suitable position to begin these steps. Thus, this grasping routine is developed as a primative which will use control parameters specified by a higher level system. Hardware difficulties (described in Section 6.1) made it possible only to implement the first two steps in the Testbed. The following sections describe the control strategies used during the first three steps of the grasping operation.

5.2 Approach

The Approach means the motions of the Puma Arm bringing the fingers into contact with the grapple lug. The robot is brought into an approach position using position control. The trajectory planner and the collision detection systems currently in available in the testbed are used to plan these motions. The approach point typically puts the fingers directly above the lug so that only motion in the finger Z direction is necessary to perform the approach. The approach point was arbitrarily chosen to be 5 cm above the grapple lug frame in this study. The mode of force control used to perform the Approach Motion is Translational Position Accommodation as described in the Section 4.2. The Approach Motion ends upon contact with the grapple lug at which time the forces and torques are saved in memory for use in the alignment step.

5.3 Alignment

When the fingers first contact the grapple lug, the contact may occur in a plane, line, or point. Assuming a point contact, this point can be used to define a new axis useful in our surface alignment algorithm. The following section discusses how this point is identified, then the next section discusses how it is used to level the fingers with respect to the grapple lug surface.

5.3.1 Point of Contact Identification

When lowering a plate onto a point contact, it is possible to use the force-torque data to locate the point of contact. Figure 5-1 illustrates this contact scenario. The frame in which the force-torque data is calculated is defined a priori to be located at the intersection of the bottom surface of the plate and the robot end-effector Z-axis with the same orientation as the end-effector frame. This sensing frame is called **S** and the point of contact is labelled P_c in Figure 5-1. A vector $\vec{p_c}$ is defined between **S** and P_c. Since P_c is located in the X-Y plane of **S**, the Z component of $\vec{p_c}$ is zero. The vector $\vec{F_c}$ describes the forces induced by contact between the point contact and the plate. The force-torque sensor resolves these forces into three forces, $\vec{F_c} = [F_{cx}, F_{cy}, F_{cz}]^T$, and three torques, $\vec{T_c} = [T_{cx}, T_{cy}, T_{cz}]^T$, with respect to the coordinate frame **S**. Looking at the forces and torque about the origin of the **S** frame, we see that

$$\vec{T_c} = \vec{p_c} X \vec{F_c} \tag{5.1}$$

where $\vec{T_c}$ is a vector of torques about the **S** frame, and $\vec{F_c}$ is a vector of forces in the **S** coordinate frame.

The forces F_x and F_y will cause no torque about the X and Y axis of **S** as they lie in the X-Y plane. The only component of this force vector which causes torques about the X or Y axis is the F_{cz}. Therefore, using the equations

$$T_x = p_y X F_z \tag{5.2}$$

$$T_y = p_x X F_z \tag{5.3}$$

we can solve for p_x and p_y which are simply the moment arms of the contact torques about **S**. This analysis only holds for point contacts between the plate and a surface. The readings for a line or a plane contact will be ambiguous. Therefore, to assure this sequence of motions can be completed it may be desirable to tilt the fingers to insure a point contact.

5.3.2 Surface Alignment

To achieve a planar contact between the grapple lug and the fingers, we must align their X-Y planes. As discussed above, if the contact occurs in a point, this point can be located. The fingers must rotate about this point to have the surface of the fingers come into planar contact with the surface of the grapple lug. To perform rotations with respect to this point, a homogeneous transformation describing the spatial relationship between the robot end-effector frame, **S**, and the point of contact, P_c, is concatenated to the equation describing the position and orientation of the frame **S**. As discussed in Section 6.4, the forces are transformed to a frame with the origin at P_c and rotationally aligned with the frame **S**.

The vector which the robot arm will rotate about is a denoted by $\vec{u}$. This vector, $\vec{u}$ is the unit vector normal to $\vec{p_c}$ which lies in the X-Y plane of **S**. The vector, $\vec{u}$, is used in the rotational position accommodation algorithm. The motion will continue about each axis until a user specified tolerance is exceded. Then, if for example, the tolerance is exceded about the X axis, the rotation about this axis will automatically terminate, but the rotation will continue about the Y axis. The same will be true for the motion about the Y axis if the threshold torque about the X axis is exceded first.

5.4 Inward Finger Motion

Using existing algorithm within the gripper controller subsystem supplied by LORD Corporation, the fingers are independently servoed and capable of both position and force control. They perform a grasping motion in two steps. First, a position move closes the fingers until a specified trip-force is reached. Then, the fingers start servoing on force until the desired force is sensed to within user-specified tolerances. At this time, the fingers lock into position.

As the gripper fingers close, forces and torques are induced in the directions in which the fingers must aligning in order to mate with the grapple lug. The force-torque data is used to control the robot arm. Specifically, Direct Force Feedback (as described in Section 4.5) changes the position of the arm in the directions in which the gripper must align proportionally to the force felt in that direction. Assuming the first two steps have been successfully completed, the alignment about both the X and Y axis to have already been performed. Also, the motion in the X direction is accomplished through the inward motion of the fingers (as described above). Motion in the other degrees of freedom the other three degrees of freedom should proportionally to the induced forces. The motion which we would like each of these degrees of freedom to track is dependent on the lug geometry. The amount which we want to move in the Z direction is the same as the amount we moved by the fingers in the X direction. Since it is possible to calculate the speed of the gripper motion, an approximate value for the gains can be found. This is important since the gripper and the robot are separately controlled and their motions must be synchronized.

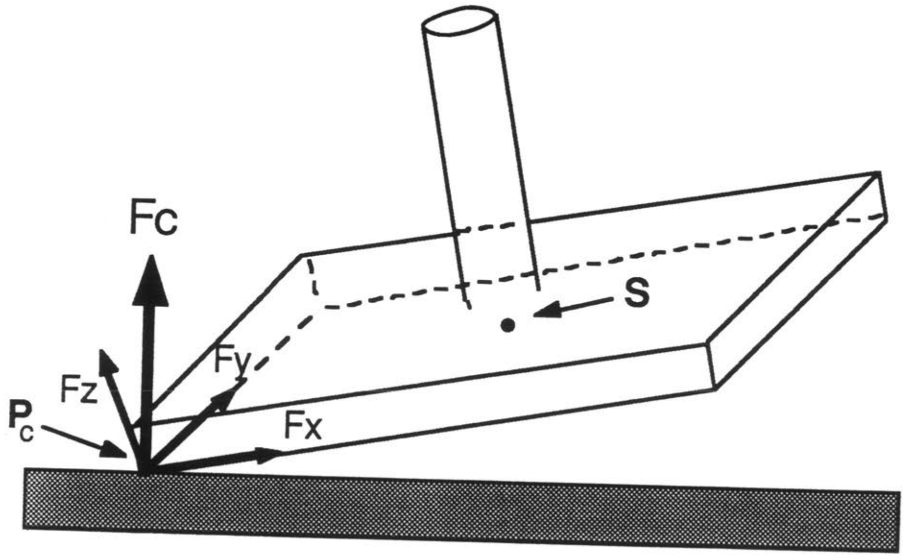

Figure 5-1: Point of Contact

Chapter 6

Implementation

Parts of the previously described grasping operation were implemented in the Telerobot Testbed. In this chapter, several issues specific to this system configuration will be discussed. First, the various components of the Telerobotic Testbed will be described. Then, the models of the robot world and the various frames used to control the robot will be specified. The robot control software is described in more detail, and the sensory information processing is discussed. A discussion of the data collected in the lab follows to highlight the issues which remain to be addressed.

6.1 System Configuration

6.1.1 Introduction

This study makes use of hardware of the Tele-Autonomous Systems Group testbed facility. A PUMA 560, six degree of freedom robot manipulator is used to perform the large scale motions involved in positioning the gripper. The robot is capable of both Joint Space and Cartesian Motions. A LORD Industrial Automation Servo-Gripper attached to the end-plate of the PUMA arm will be used to perform the grasp. Force-Torque sensors in the base of the gripper fingers provide six axis force feedback which can be used to control of the system. The gripper has independently servoed fingers which are capable of both position and force control. For this study, a LORD Wrist Force-Torque sensor and a pneumatic gripper were used in place of the Gripper Subsystem because of difficulty in integrating it with the telerobot testbed.

Specifically, the Testbed real-time control system has a sample period of 28 ms. About 20 ms of this time was free to read the force-torque data. Unfortunately, the force-torque data was only accessible from the gripper control system every 50 ms. It was decided that slowing down the control system would degrade the robots performance considerably. The changes to the gripper will include access to all six force-torque readings within the robot controllers sample period. Even though this project didn't actually use the LORD gripper, I include a description because certain assumption about the gripper capabilities were made throughout this study. The following gives a brief description of the capabilities of the gripper during a grasping motion. Changes presently being made in the gripper software will make complete integration possible in 1989.

6.1.2 LORD Instrumented Gripper

The LORD gripper subsystem was chosen for this task because many of the control parameters are user specified allowing fine-tuning of the grasping operation control. The following paragraph describes some of these capabilities. The speed is user specified during position controlled motion. The component of force or torque to use in the force control can be set. The *trip-force* signals the end of position control and the beginning of force control. The *threshold* specifies the middle range of the force to terminate the motion on for both fingers. A *balance* is set to compensate for the natural imbalance of the sensors, readings from one sensor are decreased by half the balance while readings from the other are increased by the same value. The *tolerance* gives the maximum allowable deviation from the threshold. By altering the coefficient of the *proportional gain* during force servoing, the user controls the increment of finger movement per difference in force. The rate of accumulation of errors is specified by the coefficient for the *integral gain.*[9] These capabilities allow a large variation in the control during grasping. The optimal set of parameters will probably be determined by a combination of analysis and experimentation.

6.1.3 LORD Wrist Force-Torque Sensor

A LORD Wrist Force-Torque Sensor and pneumatic gripper were used in place of the previously described LORD Instrumented Gripper. The force-torque sensors used in the Lord Wrist Sensor are strain gauges in a Maltese Cross configuration[10]. This configuration consists of 8 strain-gauges mounted to metal beams. These beams lie between two plates in the sensor. Thus, forces applied to the fingers will cause these plates to move with respect to one another causing the beams to deform. The deformation produces a voltage difference in the strain gauges. This analog strain-gauge data is digitized in a preprocessor. Calibration of the readings occurs in the force-torque sensor controller. A 6X1 vector of the forces and torques is obtained by multiplying the vector of strain gauge readings by a 6X8 calibration matrix. The components of the 6X1 force vector are orthogonal in the X, Y, and Z directions of the gripper fingers. Forces are given in Force Torque Units (1 FTU = 5 oz).

6.1.4 Telerobot TestBed Control Structure

Figure 6-1 illustrates the control structure. At the highest level, there is a Microvax running a UNIX kernel modified to permit real-time operations. Using a Robot Control C Library (RCCL), the VAX communicates over a parallel port with a Unimation Robot Controller. The Unimation Robot Controller runs a real-time communication and control language called MOPER which interrupts the VAX every 28 msec to receive commands to move the robot joints, open and close the gripper, and update the VAXS knowledge of the sensory information and robot joint position. Commands are interpreted and used to control the robot joint processors and the gripper controller. The force-torque sensor controller processes raw strain-gauge readings into calibrated force-torque information.

6.2 Frames

Each object within the testbed robots workspace has an associated frame which describes its location with respect to the base of the robot. Homogeneous transformations [11] describe the spatial relationship between the frames attached to each of the objects. The robot end-effector also has an associated frame, the location of this frame is described by an homogeneous transformation which is a function of the robots joint angles. The joint angles necessary to achieve a particular position can be calculated and used to control the position of the robot end-effector with respect to the base and consequently with respect to any object within the robots workspace. The coordinate frames are of interest during the grasping operation are the grapple lug coordinate frame and the finger coordinate frame. In the following, the method of locating these frames is discussed.

The origin of the end-effector coordinate frame is located at the last joint axis in the Puma. A new frame is centered on the bottom surface plane of the fingers is used in both the position and the force control. Section 6.3 describes how the control software determines the position and motion of this frame. The force-torque sensor also has an assosiated frame which is located in the cented of the sensor. The force-torque readings are readings are taken with respect to this coordinate frame. Force transformations between frames are discussed in Section 6.4.

All of the EOS mockups have associated frames. These frames are interconnected in a hierarchical structure forming an up-to-date model of all objects in the robots environment. The spatial relationship between the frame associated with the grapple lug and a frame associated with the ORU is stored in the model. When the grasping primative is used by the higher level control system, the positioning of the arm above the grapple lug will be based on this model.

6.3 Robot Control Language

The robot control language used in the Telerobotic Testbed is RCCL [7]. In RCCL, there is a planning level and a control level. The planning level is a 'C' program(s) which incorporates a library of robot control commands. The control level is a 'C' program consisting of real-time control algorithms which are evaluated every sample period. To control the robot, a desired set of joint angles are sent to the robot joint servo controllers each sample period.

The equation used to compute the desired set of joint angles during each sample period is

$$\mathbf{B} \cdot \mathbf{T_{6init}} \cdot \mathbf{E} \cdot \mathbf{\Delta} = \mathbf{B} \cdot \mathbf{T_6} \cdot \mathbf{E} \tag{6.1}$$

where $\mathbf{B}, \mathbf{T_{6init}}, \mathbf{E}$, and $\mathbf{\Delta}$ are 6x6 homogeneous transformation matrices. In Equation 6.1,

- $\mathbf{B}$ represents the spatial transformation between a fixed world coordinate frame and the base coordinate frame of the robot.
- $\mathbf{T_6}$ represents the spatial transformation between the base coordinate frame of the robot and the robot end-effector. It is a function of the robot joint angles.
- $\mathbf{T_{6init}}$ represents the initial spatial transformation between the base coordinate frame of the robot and the last link coordinate frame.
- $\mathbf{E}$ repesents the spatial transformation between the robot's last link coordinate frame and the end-effector coordinate frame.
- $\mathbf{\Delta}$ represents the spatial transformation between the initial end-effector coordinate frame and a coordinate frame attached to the end-effector.

Initially, Δ is the identity matrix. Adjusting Δ changes the set of joint angles computed each sample period from Equation 6.1. In this implementation, Δ is a function of the vectors $\vec{p_i}$ and $\vec{r_i}$ as determined in Chapter 4.

6.4 Sensory Information

6.4.1 Introduction

The forces used in the control algorithms must be in referenced to a particular frame which is not necessarily the force-torque sensing frame. The first part of this section discusses force transformations. In addition, the control algorithms assume that all the forces read by the sensor are induced by contact. Other forces may be present such as:

gravitationally induced loads and transient dynamic loads on the sensor. Methods of eliminating (or reducing the effect of) these extraneous readings are presented in the second part of this section.

6.4.2 Force Transformations

The force-torque readings obtained from the finger sensors are measured with respect to a coordinate frame whose origin corresponds to the center of the sensors with the same orientation as the finger frame, as described in the Section 6.2. A Jacobian, which is a function of the position vector and rotation between the two frames, is used to transform the between the frames:

$$\vec{F} = (\mathbf{J}^T)^{-1}\vec{Q} \tag{6.2}$$

$$\mathbf{J} = \begin{bmatrix} n_x & o_x & a_x & 0 & 0 & 0 \\ n_y & o_y & a_y & 0 & 0 & 0 \\ n_z & o_z & a_z & 0 & 0 & 0 \\ (pXn)_x & (pXo)_x & (pXa)_x & n_x & o_x & a_x \\ (pXn)_y & (pXo)_y & (pXa)_y & n_y & o_y & a_y \\ (pXa)_z & (pXo)_z & (pXa)_z & n_z & o_z & a_z \end{bmatrix} \tag{6.3}$$

$\vec{Q}$ is a 6X1 vector of the generalized measured forces and $\vec{F}$ is a 6X1 vector of the generalized interaction forces. And, in the Jacobian matrix ($\mathbf{J}$), n, o, a, and p are the components of a homogeneous transformation describing the spatial relationships between the two frames.

The LORD Wrist Sensor control system has the capability to automatically change frames given this spatial relationship before a priori. An initial sensing frame for the grasping operation is defined in this manner to be on the bottom surface of the fingers. During the grasping operation, the transformation, **J**, is used in the control software to perform an on-line frame change (see programs in Appendix A).

6.4.3 Sensor Biasing

The force-torque sensors measure the load on the robot end-effector. The sensor, the gripper, and the fingers contribute partially to this load because of the forces gravity exerts on them. The effect of this loading will be reduced in orbit, however, since demonstration grasping operations will be performed in the laboratory, the effects of gravitational loading on the force-torque readings must be examined.

In order to obtain accurate reading of the forces and torques generated by contact between the fingers and their environment, it is necessary to eliminate the components corresponding to the weight of the gripper and fingers. The process of eliminating these readings to within the force-torque sensors resolution is called biasing the sensors. To bias the sensors, the force-torque readings at all subsequent readings are calculated relative to the initial readings.

6.4.4 Transient Dynamic Loading

Motion of the arm creates loads on the wrist sensor. When the robot first begins a trajectory from a stationary position, the entire robot inertia must be accelerated. Consequently, there will be a reaction force to this acceleration in the opposite direction as the motion. Also, as the robot moves at a constant velocity transient dynamic loads are induced by vibrations in the arm. The magnitude of these loads are quantified in 6.5.

6.5 Performance Evaluation

The performance of the first parts of the control strategy described above were evaluated in the Tele-robot testbed. The forces from the wrist sensor and the motion of the robot end-effector were monitored. The programs written to perform the first steps of the grasping operation are contained in Appendix A. The programs allow the user to input various control parameters such as starting position, driving increments, and motion termination thresholds.

The following sections describe various control issues and illustrates them with data taken in the Testbed. To collect this data portions of the code in Appendix A were eliminated or modified. Force-torque data was produced by storing force values to an array during robot motion in the RCCL control level. This data was written to a file as the last step in the programs. Graphs were generated from the datafile using a two dimensional plotting program called Plot2d.

6.5.1 Approach Motion

Filtering–

A filter is used to eliminate the effects of transient dynamic loads on the LORD Wrist Force Torque Sensor. The wrist sensor has the software capability to perform a window smoothing operation where the size of the window is user specified. The Wrist Force-Torque Controller operates at 104 Hz. Because of the processing time needed for the filtering operation, the FT Controller only operates at 83 Hz when the filter is activated. The LSI-11 only up-dates the force and torque information at 35.7 Hz. The need for this filtering function is illustrated by the Figures 6-3 and 6-4 which show the forces on the robot end-effector in the Z direction with no filtering and with the filtering option activated, respectively. In Figure 6-4, the window size was 4 and the speed of the arm was 3.6 mm/sec. As can be seen from the figures, the spikes of the unfiltered signals more than double the spikes of the filtered signals. These tests were performed at a relatively slow speeds, at higher speeds, the spikes can approach our motion termination threshold of 50 FTU when the filter is inactive (see Figure 6-5).

ForwardDrive–

To determine an optimum set of parameters during the Translational Position Accommodation, tests were run varying the driving increment (the distance the robot moves during each sample period). The filter described above was active during these tests. It was learned that a large driving increment caused large force values to be generated during the motion in free space. For example, at a driving increment of 18 mm/sec, the magnitude of the force readings was as great as 9 FTU's (see Figure 6-6). The large variations in force are thought to be caused by the acceleration experienced by the robot arm and the vibration of the motors in the robot joints. Also, in Figure 6-6, the reaction force to the initial acceleration is distinguishable (the large negative spike). A driving increment of 3.6mm/sec which keeps the noise to within 3 FTU's was selected for the rest of the tests (see Figure 6-7).

6.5.2 Instantaneous Contact Forces

As described in Chapter 4, position accommodation was used when bringing the end-effector in contact with a surface. The idea behind this was to reduce the large overshoot of the threshold force. The overshoot was 500% for relatively slow robot speeds (3.6 mm/sec). The overshoot, of course, increases for higher speeds. The use of a position accommodation only reduced the overshoot marginally (see Figures 6-8 and 6-9). These curves show the robot pushing against the environment until a threshold force of 50 FTU's was reached. At this time the motors stop driving the robot forward. However, because the robots inertia cannot be halted instantaneously, the forces overshoot the threshold value. Data after the maximum force was created by a programming bug which didn't stop execution and can be ignored. Figures 6-8 and 6-9 show that the position accommodation for motion act in this environment almost like guarded motions.

6.5.3 Point of Contact Identification

As discussed in Section 5.3.1, it is theoretically possible to find the point of contact between the flat surface of the fingers and the grapple lug. However, this method may be imprecise for the following reasons: the contact between the two surfaces will not be exactly a point and the rigid body model of the arm may not be accurate during contact because of compliance in the robot arm.

The point of contact identification algorithm was tested experimentally in the Telerobot Testbed to see if it generates information about the point of contact sufficiently precise to be of use in the surface alignment procedure. For this experiment, a special tool as shown in Figure 6-10 was attached to the end-effector of the robot. The tool was lowered onto an aluminum pin which protruded from a block within the robots workspace. The exposed end of the pin was tapered to assure a point contact. Joint angles to enable the robot to achieve a configuration above the pin were obtained by aligning the tool with the corners of the block then bringing the center of the plate into contact with the pin. The software written to evaluate the point of contact identification algorithm allowed the user to specify the starting distance above the pin, and the X and Y position of the center of the plate relative to the pin.

Motion towards the pin was accomplished using Translational Position Accommodation as discussed in section 4.2. The motion was stopped when contact between the plate and the pin produced a force value equal to the threshold of 50 FTU's. The resulting static force-torque readings were used to calculate the point of contact. Table 6.1 shows the performance of the point of contact identification algorithm. In this table, point A was nominally at the center of the plate, $(0,0,0)^T$. Point B was nominally 30 mm from the center of the tool in the X direction, $(30.0,0,0)^T$. Point C was nominally 60 mm fron the center of the tool in the X direction, $(60.0,0,0)^T$. As seen from Table 6.1, the worst case error in ten trials was 2.05 mm. The average error for the X distances for Point A, Point B, and Point C were 0.72mm, 0.55mm, and 0.80mm, respectively. Therefore, the magnitude of the errors were not found to be a function of the distance from the center of the plate. Table 6.1 indicates that the point of contact identification algorithm is sufficiently precise to be used by the surface alignment procedure.

	Point A		Point B		Point C	
#	X (mm)	Y (mm)	X (mm)	Y (mm)	X (mm)	Y (mm)
1	0.00	-0.79	31.50	0.79	62.05	-1.36
2	0.00	0.00	30.97	0.54	60.48	1.61
3	0.92	-1.83	30.42	-0.61	60.00	1.60
4	0.93	1.86	29.88	-0.66	60.30	-0.66
5	1.71	0.85	30.89	0.00	61.60	0.00
6	0.00	1.27	30.24	-1.21	59.96	0.83
7	-0.94	0.94	30.48	0.92	60.82	1.43
8	0.64	0.00	30.16	-0.13	61.26	-0.75
9	1.37	-0.69	30.48	-0.60	61.40	0.73
10	0.71	0.00	30.48	0.00	60.10	0.00

Table 6.1: Performance of Point of Contact Algorithm

6.5.4 Levelling

To demonstrate the entire operation, all the step thus far programmed were run as one procedure. The plate was positioned above a block on the taskboard and tilted using a teach demo program to an arbitrary angle. The plate approached the block, made contact, and rotated until level. Although no quantitative data was taken, visual observation indicated a slight drift in the vector about which the plate was rotating. This effect is thought to be caused by the change in gravitational loading on the sensor, gripper, and fingers. For small angles of rotation this effect is reduced, however, for demonstration purposes larger orientaion misalignments are desirable. The following describes how this effect can be eliminated. A method of eliminating the effect of gravity loading is presented in the following chapter.

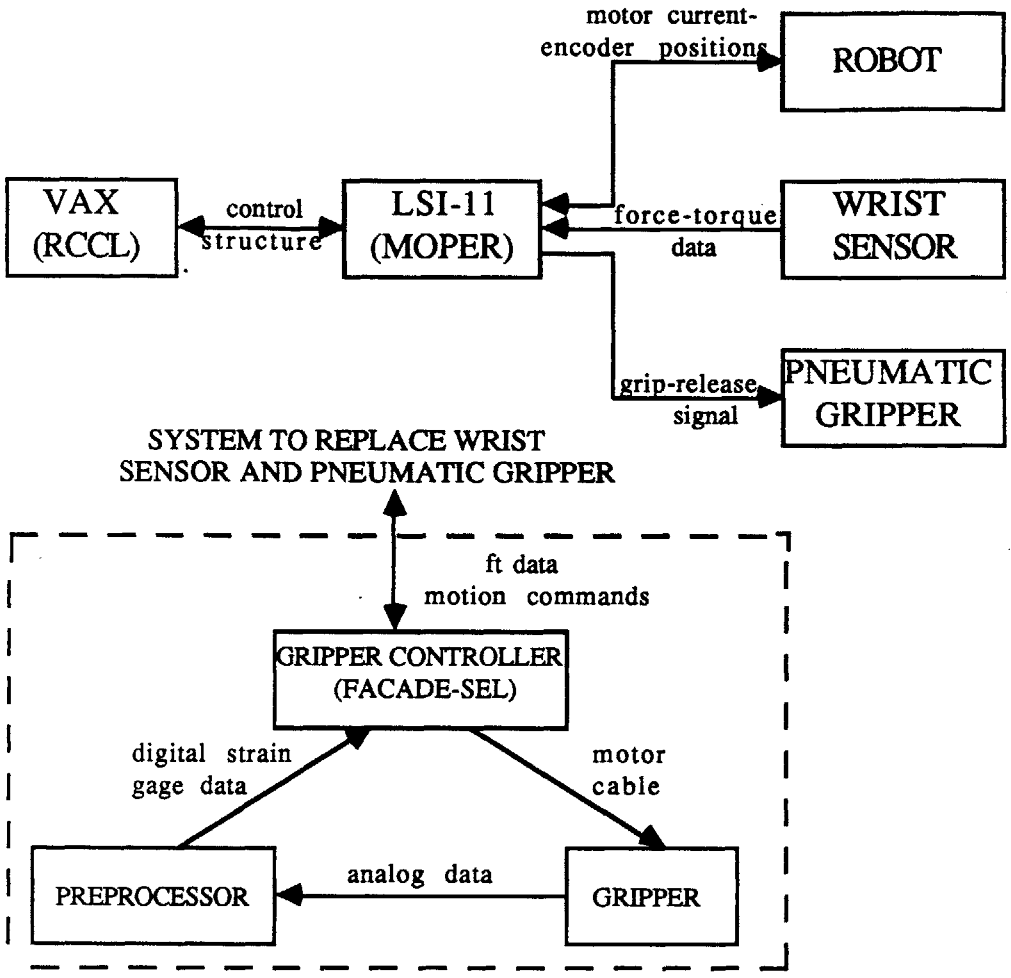

Figure 6-1: System Configuration

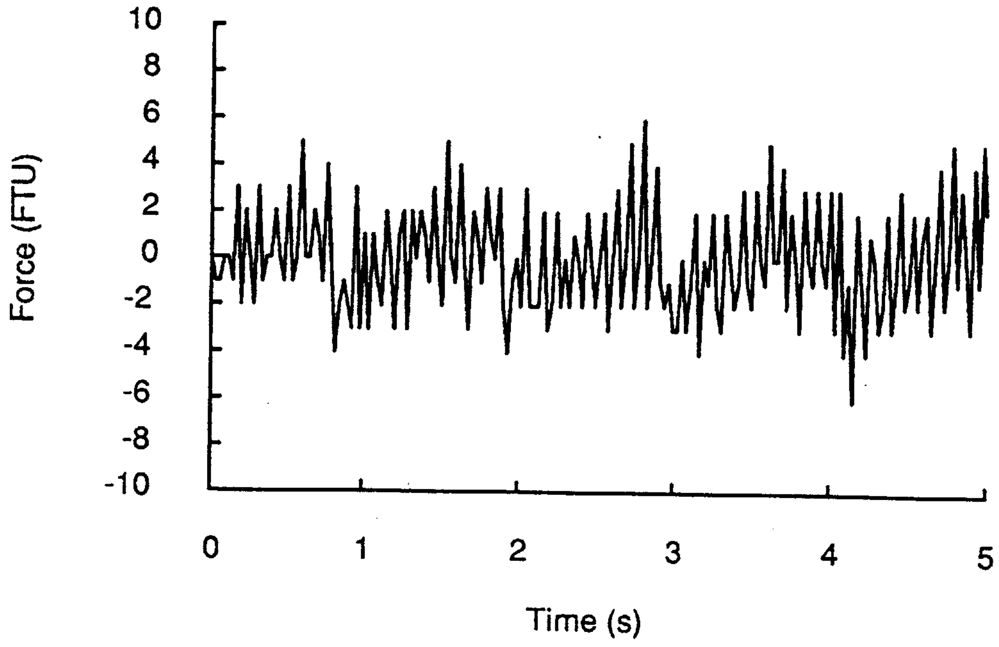

Figure 6-3: Normal Force at a Velocity of 3.6 mm/sec without Filtering

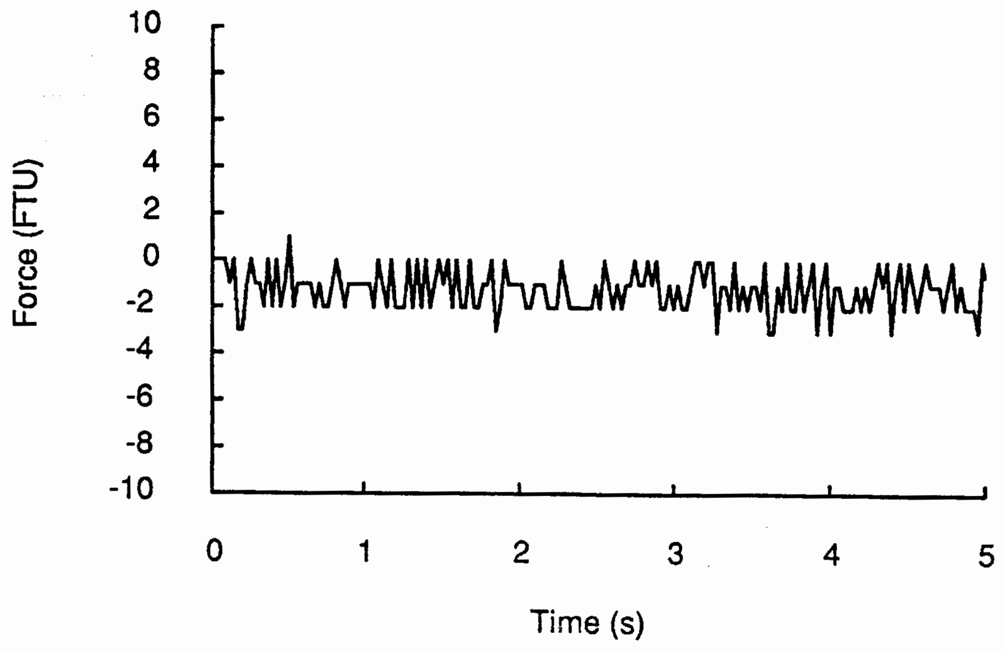

Figure 6-4: Normal Force at a Velocity of 3.6 mm/sec with Filtering

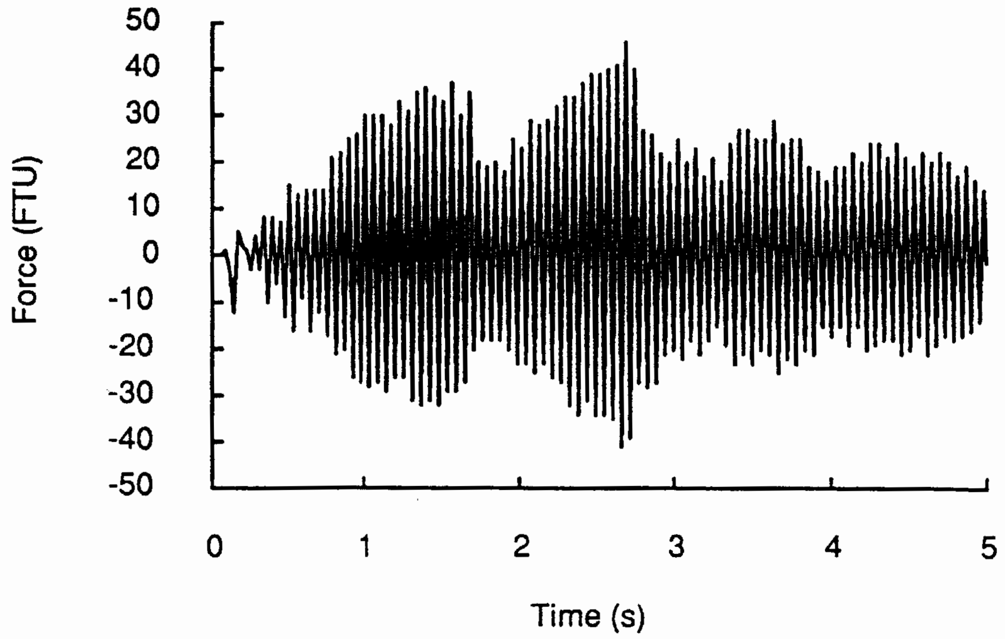

Figure 6-5: Normal Force at a Velocity of 18 mm/sec without Filtering

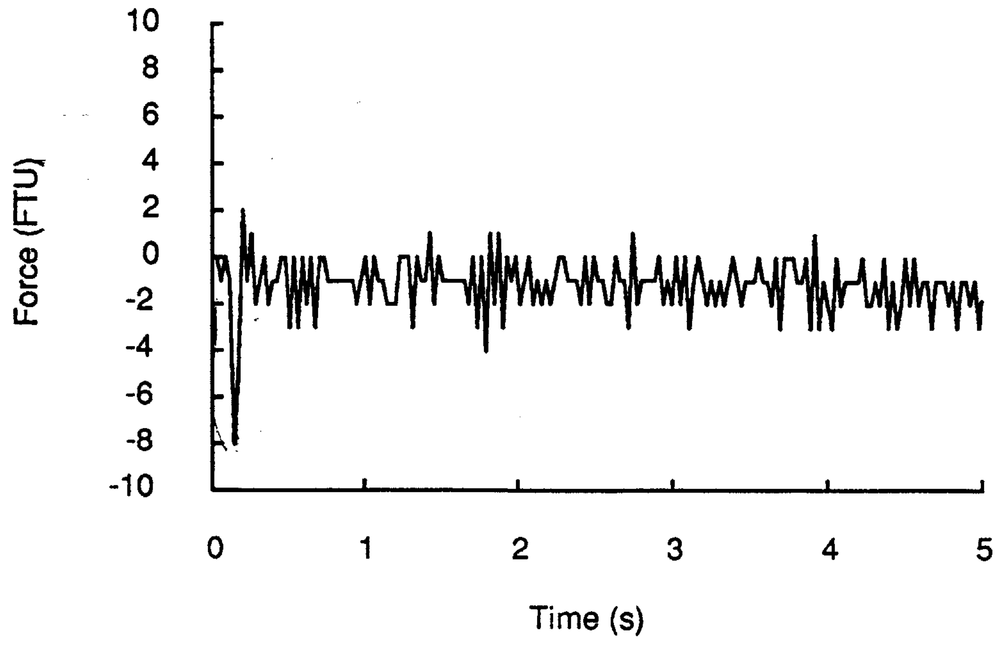

Figure 6-6: Normal Force at a Velocity of 18 mm/sec

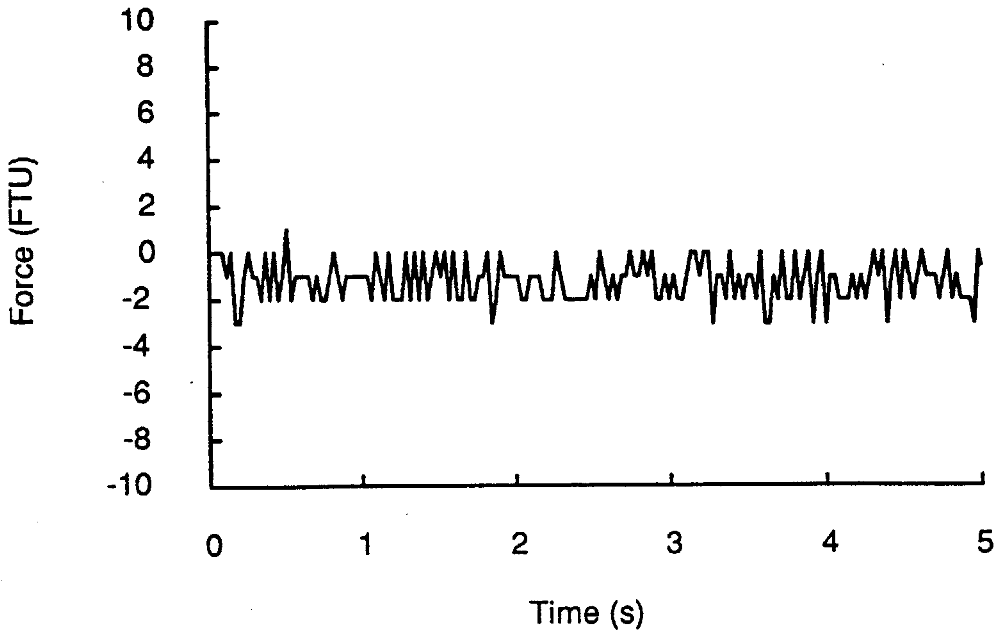

Figure 6-7: Normal Force at a Velocity of 3.6 mm/sec

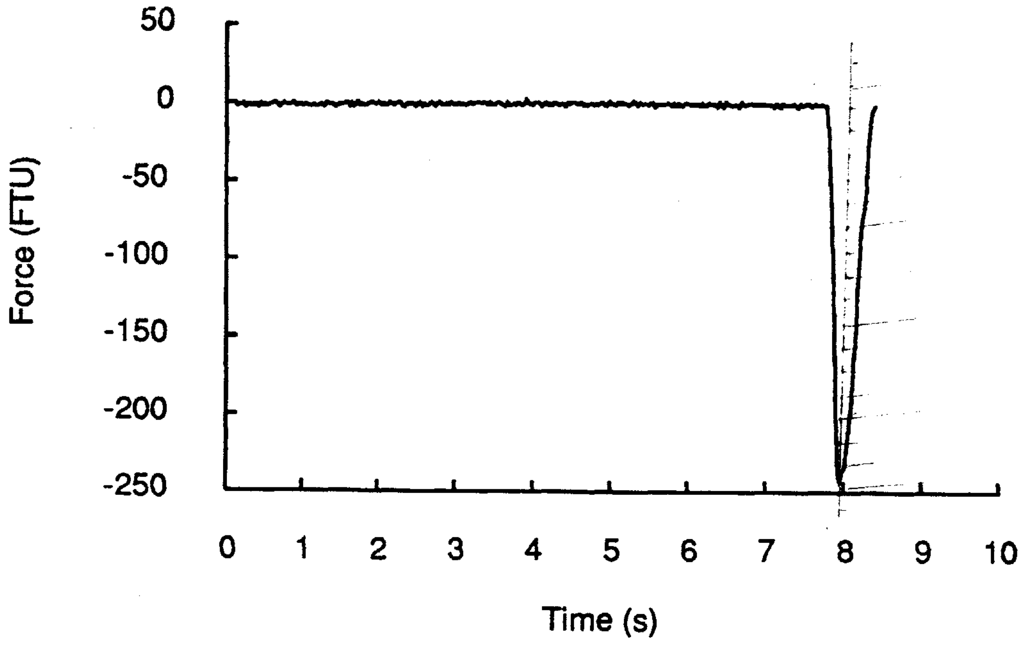

Figure 6-8: Guarded Motion; Speed = 3.6 mm/s; Threshold = 50 FTU's)

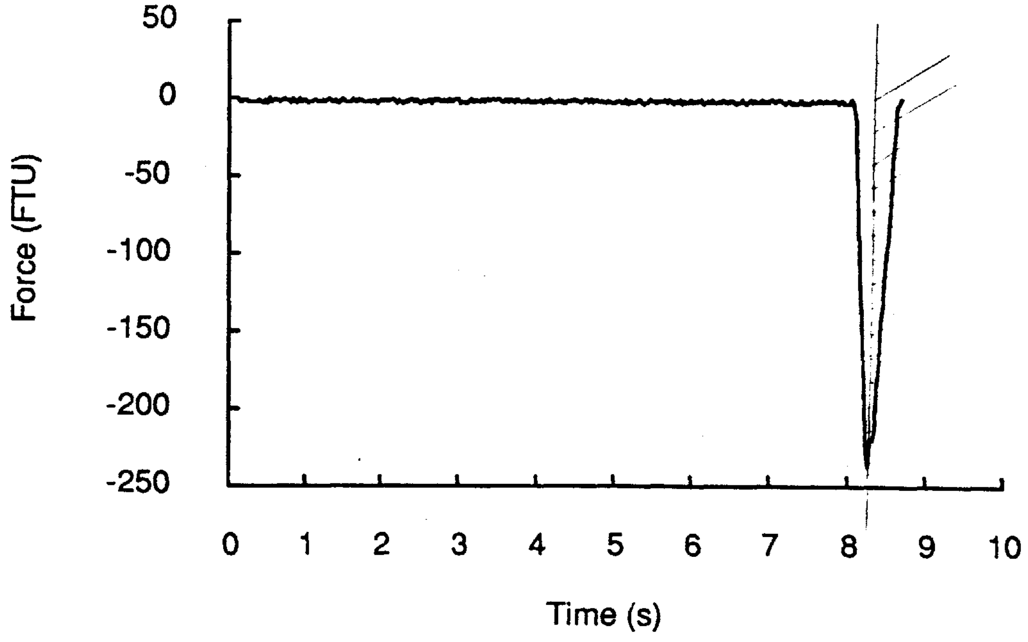

Figure 6-9: Forward Motion Posision Accommodation; Speed = 3.6 mm/s; Threshold = 50 FTU's

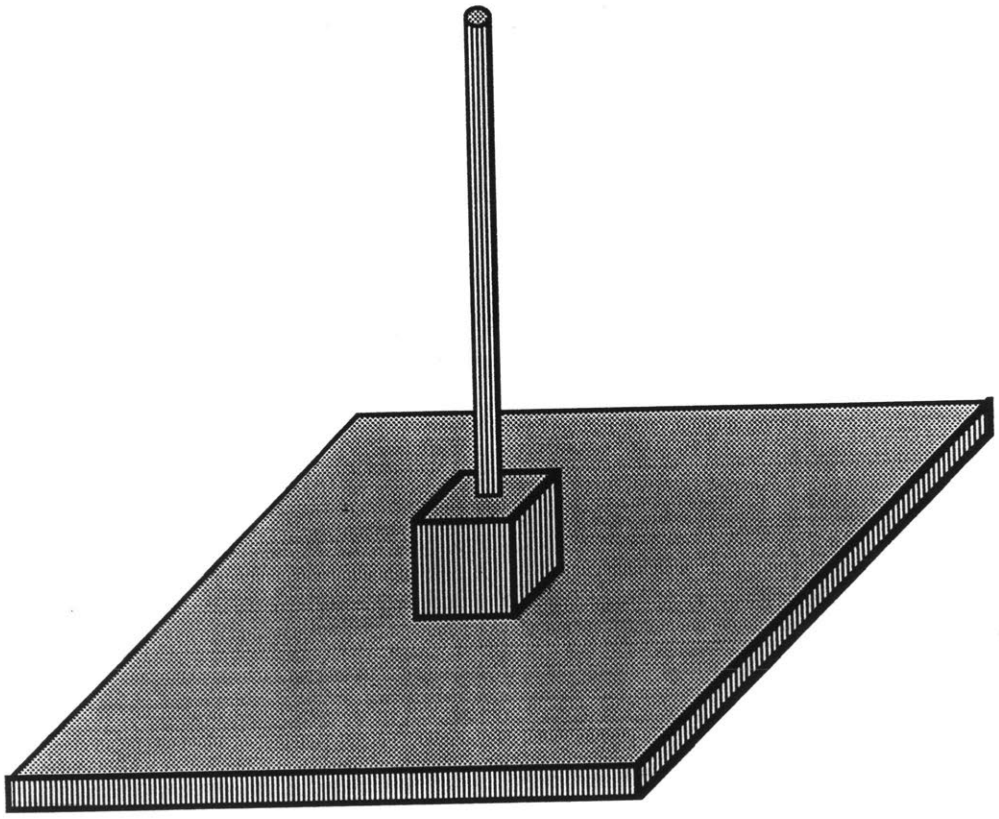

Figure 6-10: Special Tool used to Implement Steps of Grasping Operation

Chapter 7

Conclusions and Recommendations

A strategy for grasping an ORU has been developed. Hardware integration problems prevented the implementation of the entire grasping operation. Various components of the entire grasp were tested using a special tool and the LORD Wrist Force-Torque Sensor. Results to date show that implementing a grasp is feasible using a six degree of freedom arm.

Experimental data indicates that the effect of manipulator inertia on sensor readings may degrade any real-time fedback control. When the arm first begins a motion, there is a reaction force read by the force-torque sensors. Also, the force-torque readings vary as the arm moves at a constant speed through free space. Performing motions at slower speeds and smoothing the force-torque readings at a higher rate than the robot sample period reduces these effects considerably.

The data also indicates that using position accommodation instead of a guarded motion produces only a slight improvement in performance. However, the problem of overshoot upon contact still remains. Several solutions to the overshoot problem are possible. Reducing the threshold to allow for the overshoot is the simplist. However, since the percent overshoot is a function of robot speed, complience of environment, and sampling rate, the maximum force felt will vary. Keep in mind that a low threshold can be triggered by the disturbances described above. Proximity sensors on the fingers could be used to almost halt the robot before contact. This solution involves adding additional hardware and integrating more control data. Alternately, it is felt that the performance of the position accommodation algorithm described above would be greatly improved by attaching a rubber pad to the fingers where contact occurs. The contact forces would build up gradually, and the position accommodation algorithm would halt the inertia of the robot before the threshold force value was reached.

The results of the point of contact identification look promising. Using the force-torque information to locate objects may have other useful applications such as indicating how a tool held by the robot is contacting a surface. The readings seem biased in the positive direction. A more precise method of aligning the plate with the taskboard may improve the readings.

Visual observations show a drift in the point of rotation in the levelling operation. This is thought to be caused by gravity loading. The problem with automatic biasing as described above is that once the sensors are biased for a initial gripper orientation, any deviation from this initial orientation will induce changes in the readings from the wrist sensor. However, assuming that the orientation of the robot end-effector with respect to the gravity vector and the location of the center of mass of the sensor, gripper, and fingers are known, the gravitationally induced forces can be calculated, transformed to the sensor frame, and subtracted from the force readings. In RCCL, this transformation would be relatively simple to program because the angle between the gripper the gravity vector is accessible from memory. Lack of time prevented implementation.

Areas which require further work include lug/finger design and error detection and recovery. The lug/finger design results from modifying the standard grappling lug for robotic applications while retaining physical similarity. A better approach might be to design to maximize some parameters such as positioning error tolerance or area of contact. A more symmetric design such the one shown in Figure 7-1 would distribute the positioning error tolerance equally in and about the X and Y axes. Also, it would increase the positioning error tolerances in the Z direction and about the X and Y axes. In addition. the prototype design simplifies the error detection and recovery work planned by the Tele-Autonomous Systems Group. With the existing design the number of possible contact scenarios (contact between a distinct surface on the lug and a distinct surface on the fingers forms one scenario) is greater that 50. Before further work is done in these areas the mechanical design should be modified.

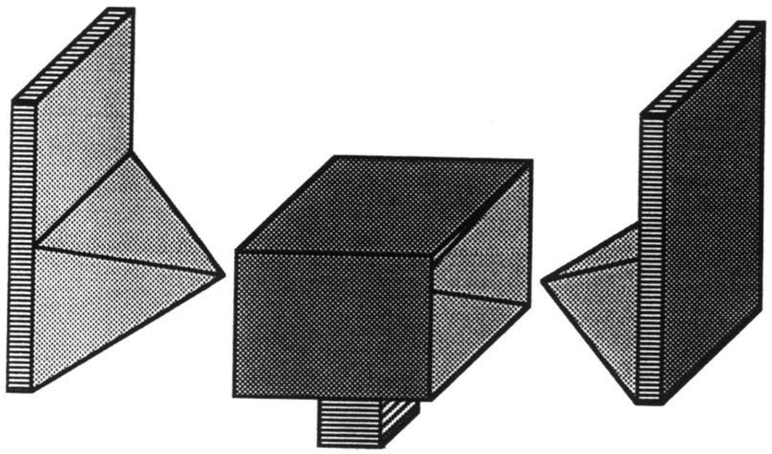

Figure 7-1: Prototype Grapple Lug and Finger Design

Bibliography

[1] TAS Group *Polar Platform Robotic Servicing Evaluation: FY 88 Laboratory Research Results.* JPL, 1986.

[2] Daniel E. Whitney *Force Feedback Control of Manipulator Fine Motions* ASME J. Dyn Sys., Meas., Contr.:91-97., 1977.

[3] J. K. Salisbury *Active Stiffness Control of a Manipulator in Cartesian Coordinates.* Proc. 19th IEEE Conf. on Decision and Control, 1980.

[4] M. H. Raibert and J. J. Craig *Hybrid Position/Force Control of Manipulators.* Transactions of the ASME, 1981.

[5] Matthew T. Mason *Compliance and Force Control for Computer Controlled Manipulators.* MIT Press, 1981.

[6] Daniel E. Whitney *Historical Perspective and State of the Art in Robot Force Control.* International Journal of Robotics Research, 1987.

[7] Vincent Hayward *Robot Manipulator Control under Unix RCCL: A Robot Control "C" Library.* The International Journal of Robotics Research, 1986.

[8] Richard P. Paul *Robot Manipulators: Mathematics, Programming, and Control.* MIT Press, 1981.

[9] LORD Automation Division, *Instrumented Gripper Operating Manual* April 1988, pgs 5-11.

[10] Antal K. Bejczy, *"Smart Hand" - Manipulator Control Through Sensory Feedback.* JPL, 1983, pg 30.

[11] John J Craig *Introduction to Robotics- Mechanics and Control.* Addison-Wesley Publishing Company, 1986.

APPENDIX

Programs for Implementing Grasping Operation

Parameter Initialization Macro

```
/**A macro to set the parameters in find_rot.c grasping program.
 ** Loop after return from find_rot.c to restart.
 ** BUGS--hazardous to restart, sometimes crashes system.
 ** Suggest - check initialization of all parameters **/

#include <signal.h>
#include <stdio.h>
#include "/projects/robot/rccl/h/rccl.h"
#include "/projects/robot/rccl/h/rci.h"
#include "/projects/robot/rccl/h/macros.h"
#include "/projects/robot/rccl/h/fsense.h"
#include "RTC.h"
#include "/projects/robot/rccl/h/hand.h"


    int         acceltime;
    int         segtime;
    extern real for_drive;
    extern real back_drive;
    extern real rot_drive;
    extern char average[];


main(argc, argv)
    int             argc;
    char          **argv;
{
    TRSF_PTR        gentr_jntm();
    TRSF_PTR        b, z;
    POS_PTR         p0;
    char           *pose = "ldf";
    JNTS            jnts;
    double          rng[6];
    struct findrotCmd d;
    char          not_done[3];
    char          resp[6];
    float       r1, r2, r3, r4, r5, r6; /*Joint Angles for a */
                                        /*starting position  */


    if (rccl_open(argc, argv) == -1) {
        printf("rccl_open\n");
        exit (-1);
    }
    if (rccl_control() == -1) {
        printf("rccl_control\n");
        exit (-1);
    }

    not_done[0] = 'n';

        printf("Enter x,y,z components for direction vector<0.0 0.0 1.0>:\n");
        scanf("%f %f %f", &d.dir.x, &d.dir.y, &d.dir.z);
        printf("Enter distance in (mm)<300.0>:\n");
        scanf("%f", &d.maxdist);
        printf("Enter acceleration time in (msec)<1000>: \n");
        scanf("%d", &acceltime);
        printf("Enter segment time in (msec)<180000>: \n");
        scanf("%d", &segtime);
        printf("Enter for_drive in (mm)<0.1>: \n");
        scanf("%f", &d.for_drive);
        printf("Enter back_drive in (mm)<0.0001>: \n");
        scanf("%f", &d.back_drive);
        printf("Enter rot_drive in (mm)<0.0001>: \n");
        scanf("%f", &d.rot_drive);
        printf("Average F/T readings [y/n]<y> ?\n");
        scanf("%s", average);
        printf("Vector of Joint Angles for Start: \n");
        scanf("%f %f %f %f %f %f", &r1, &r2, &r3, &r4, &r5, &r6);
        printf("%f %f %f %f %f %f\n", r1, r2, r3, r4, r5, r6);

/* The following are the joint angles necessary to bring the plate tool*/
/* approx 10 cm above the point contact which sits in the reamed block*/
/* on the task board */
/* Comment out the last question when using these values */
/*      r1 = -51.61;
        r2 = -103.36;
        r3 = 2.24;
        r4 = 0.0;
        r5 = -78.88;
        r6 = 38.39;  */

/* Or to approach reamed block corner from an angle*/

/*      r1 = -43.17 */
/*      r2 = -107.26*/
/*      r3 = -11.08*/
/*      r4 = -7.54*/
/*      r5 = -62.99*/
/*      r5 =  50.28*/

        printf("%f %f %f %f %f %f\n", r1, r2, r3, r4, r5, r6);

    while(not_done[0] == 'n'){
  b = gentr_jntm("B", r1, r2, r3, r4, r5, r6, pose);
        p0 = makeposition("P0", t6, EQ, b, TL, t6);
        setvel(30, 30);
        setmod('j');
        setconf(pose);
        move(p0);
        move(p0);
        waitfor(completed);

/*      OPEN;*/   /*Commented out when tool already in to avoid dropping*/
        printf("Press <CR> when ready to close gripper");
        scanf("%s", resp);
        CLOSE;

        d.arm = 1;
        printf("touch move begins\n");
        findrot_move(&d);
        printf("touch move finished\n");
        printf("Are you done [y/n] ?\n");
        scanf("%s", not_done);

    }           /* while */
}                               /* main */

TRSF_PTR gentr_jntm(name,q1,q2,q3,q4,q5,q6,conf)
char *name;
real q1,q2,q3,q4,q5,q6;
char *conf;
{
     TRSF_PTR t;
     JNTS jnts;
     double ang[6],rad[6],rng[6];

     jnts.conf = "   ";
     ang[0] = (double) q1;
     ang[1] = (double) q2;
```

```
        ang[2] = (double) q3;
        ang[3] = (double) q4;
        ang[4] = (double) q5;
        ang[5] = (double) q6;

        degree_to_radian(rad,ang);
        radian_to_range(rng,rad);
        jnts.th1 = rng[0];
        jnts.th2 = rng[1];
        jnts.th3 = rng[2];
        jnts.th4 = rng[3];
        jnts.th5 = rng[4];
        jnts.th6 = rng[5];

        t = newtrans(name,const);

        jns_to_tr(t,&jnts,NULL);

        conf[0] = jnts.conf[0];
        conf[1] = jnts.conf[1];
        conf[2] = jnts.conf[2];
        conf[3] = jnts.conf[3];

        return(t);
}

FT_command(code)
     int              code;
{
     SET_CMDVAL(force_sensor, FT_OFF);
     SET_CMDVAL(force_sensor, code);
     nap(1.0);
     SET_CMDVAL(force_sensor, FT_ON);
     nap(0.5);
}                                   /* FT_command */
```

Planning Level

```
 ** Planning level of first two step of grasping operation:
    locate surface and align planes.

 ** Author: Helen Greiner  (adapted from newtouchevfn2.c)  **/


#include <signal.h>
#include <stdio.h>
#include "/projects/robot/rccl/h/rccl.h"
#include "/projects/robot/rccl/h/rci.h"
#include "/projects/robot/rccl/h/macros.h"
#include "/projects/robot/rccl/h/fsense.h"
#include "RTC.h"

#define FOR_DRIVE_MODE          0
#define ROTATE_MODE             1
#define BACKDRIVE_MODE          2
#define FOR_THRESHOLD           50.0
#define SMPL                    0.028

extern float    forceValues[536][6];    /*2D array of FT data*/
extern int      scnt;               /*number of samples taken*/
extern int      find_rot_evfn();
extern real     gain;
extern short    initforce[];
extern VECT     dir;
extern real     maxdist;
extern int      evfnResult;
extern int      mode;
extern real     for_drive;
extern real     rot_drive;
extern int      acceltime;
extern int      segtime;
extern float    rx, ry, rz;
extern float    tdiff;
extern float    rotx;
extern float    roty;
extern float    rotz;
extern  float    xcontact;      /*extern should be added when*/
extern  float    ycontact;      /*bias for transformed forces*/
extern  float    zcontact;      /*is implemented*/

char            average[3];

findrot_move(d)
        struct findrotCmd *d;

{
        FILE            *fpout;
        FILE            *fopen();
        TRSF_PTR        tp;     /* adds plate to position equation*/
        TRSF_PTR        t;      /* the T6 homo. trans. in last sample*/
        TRSF_PTR        c;
        POS_PTR         pt;
        int             i;
        char            answer[6];
        float           tool_length;
        dir.x = d->dir.x;
        dir.y = d->dir.y;
        dir.z = d->dir.z;

        maxdist = FABS(d->maxdist);
        evfnResult = -1;
        mode = FOR_DRIVE_MODE;                  /*sets type Pos. Acc.*/
        for_drive = d->for_drive;
        rot_drive = d->rot_drive;
        gain = for_drive / FOR_THRESHOLD;       /*only needed to print*/
        tool_length = 162.4;                    /* in mm of course*/

        printf("gain       = %f\n", gain);
        printf("for_drive  = %f\n", for_drive);
        tp = gentr_trsl("TP", 0.0, 0.0, tool_length);
        c = newtrans("C", find_rot_evfn);
        t = newtrans("T", const);
        assigntr(t, t6);
        pt = makeposition("PT", t6, tp, EQ, t, c, unitr, TL, t6);
                        /*should include another tp in right side*/
                        /*add and test it in lab*/

        printf("Hit <cr> when ready to bias the sensor:\n");
        scanf("%s", answer);
        FT_command(FT_BS);
        FT_command(FT_SF3);      /*frame at bottom surface in center*/
                                 /*of platetool                    */

        if (average[0] == 'y') {
                FT_command(FT_EF4);
        }
        for (i = 0; i < 6; i++)                         /*probably no need */
                initforce[i] = how.forceValues[i];      /* values -2<ftu<2 */
        printf("Initial force: ");
        for (i = 0; i < 6; i++)
                printf("%d ", how.forceValues[i]);
        printf("\n");

        setime(acceltime, segtime);
        setmod('c');

        printf("t->p.x  t->p.y  t->p.z \n");
        printf("%7.2f %7.2f %7.2f \n\n", c->p.x, c->p.y, c->p.z);

        move(pt);
        completed++;
/* Screen display of force Values is eliminated to save time */
        while (completed) {
        /*      printf("%4d %4d %4d %4d %4d %4d\n",
                 how.forceValues[0], how.forceValues[1], how.forceValues[2],
                how.forceValues[3], how.forceValues[4], how.forceValues[5]); */
        }


        printf("t->p.x  t->p.y  t->p.z \n");
        printf("%7.2f %7.2f %7.2f \n\n", c->p.x, c->p.y, c->p.z);

/*Finds contact point for point contact in tool frame*/
/*Torque scale factor for this tool is 4             */
        xcontact = -(how.forceValues[4] * 4.0) / how.forceValues[2];
        ycontact = (how.forceValues[3] * 4.0) / how.forceValues[2];
        zcontact = 0.0;


        printf("x contact = %7.2f \n", xcontact);
        printf("y contact = %7.2f \n\n", ycontact);


        printf("Force Values in FT units\n");
        printf("%4d %4d %4d %4d %4d %4d\n",
                how.forceValues[0], how.forceValues[1], how.forceValues[2],
                how.forceValues[3], how.forceValues[4], how.forceValues[5]);
        /* CODE FOR BACKDRIVE*/

        if (evfnResult = 1){

        printf("Entering BACKDRIVE_MODE");
        mode = BACKDRIVE_MODE;                  /*switch type of Pos. Acc.*/
        move(pt);
        completed++;

/* screen display eliminated */
        while (completed) {
        /*      printf("%4d %4d %4d %4d %4d %4d\n",
                 how.forceValues[0], how.forceValues[1], how.forceValues[2],
                how.forceValues[3], how.forceValues[4], how.forceValues[5]);
*/
        }


        /* CODE FOR ROTATING */


        mode = ROTATE_MODE;             /* switch type of Pos. Acc.*/

/*Determines from point of contact the amount we wish to rotate in */
/*Each direction */
/*Used instead of changing frames so we can control each axis     */
/*independantly                                                   */
/* Vector of rotation is found by taking a normal to vector from  */
/* tool frame to point of contact in plane of the plate           */

        rotx = -ycontact/(sqrt(xcontact*xcontact+ycontact*ycontact));
        roty =  xcontact/(sqrt(xcontact*xcontact+ycontact*ycontact));
        rotz = 0.0;



/* Here we should calculate transformed initial-forces */
/* Should be added*/

/*experimental to test direction of rotations*/
/*      rotx = 1.0; */
/*      roty = -1.0; */
/*      xcontact = 3.0; */
/*      ycontact = 3.0; */


        printf("rotx = %7.2f \n", rotx);
        printf("roty = %7.2f \n", roty);
        printf("rotz = %7.2f \n", rotz);

        tp = gentr_trsl("TP", (xcontact*25.4), (ycontact*25.4), tool_length);
        c = newtrans("C", find_rot_evfn);
        t = newtrans("T", const);
        assigntr(t, t6);
        pt = makeposition("PT", t6, tp, EQ, t, tp, c, unitr, TL, tp);

        for (i = 0; i < 6; i++)
                initforce[i] = how.forceValues[i];
        printf("Initial force: ");
        for (i = 0; i < 6; i++)
                printf("%d ", how.forceValues[i]);
        printf("\n");

        setime(acceltime, segtime);
        setmod('c');
        printf("t->p.x  t->p.y  t->p.z \n");
        printf("%7.2f %7.2f %7.2f \n\n", c->p.x, c->p.y, c->p.z);

        move(pt);
        completed++;
/* screen print out off*/
        while (completed) {
        /*      printf("%4d %4d %4d %4d %4d %4d\n",
                 how.forceValues[0], how.forceValues[1], how.forceValues[2],
                how.forceValues[3], how.forceValues[4], how.forceValues[5]);
                printf("%7.5f %7.5f %7.5f %7.5f \n",
                rx, ry, rz, tdiff); */
        }


        printf("t->p.x  t->p.y  t->p.z \n");
        printf("%7.2f %7.2f %7.2f \n\n", c->p.x, c->p.y, c->p.z);

        printf("Force values in FT units \n");
        printf("%4d %4d %4d %4d %4d %4d\n",
                 how.forceValues[0], how.forceValues[1], how.forceValues[2],
                how.forceValues[3], how.forceValues[4], how.forceValues[5]);


}
        printf("%7.5f %7.5f %7.5f %7.5f \n",
                rx, ry, rz, tdiff);

        FT_command(FT_SF0);
        FT_command(FT_EF0);



        freetrans(t);
        freetrans(c);
        freetrans(pt);

/* open file and print stored force values into it */

while (evfnResult<5) {}

        if ((fpout = fopen ("find_rot.dat", "w")) == NULL){
            printf("can't open \n");
            exit(1);
            }


        for (i = 0; i<scnt; i++){
            fprintf (fpout, "%4d %4d %4d %4d %4d %4d %4d \n",
            i*SMPL,
            forceValues[i][0],
            forceValues[i][1],
            forceValues[i][2],
            forceValues[i][3],
            forceValues[i][4],
            forceValues[i][5]);

        }
}
                                /* find_rot.c */
```

```
 **      @(#)newtouchevfn.c       1.0      (JPL MCM)       7/18/88
 **/
static char     SID[] = "@(#)newtouchevfn.c      1.0      (JPL MCM)        7/18/88";

/**
 ** The eval function used by the find_rot grasping program
 ** It assumes the use of filtering (average of 4) in the force/torque
 ** sensors.
 ** Mode of Pos. Acc. must be set in planning level
 ** Mode can be as of now Translational Motion Position Accommodation
 **                      Rotational Motion Position Accomodation
 **                      Backdrive

 ** Author: Helen Greiner       (adapted from newtouchevfn2.c)

 **/

#include "/projects/robot/rccl/h/rccl.h"
#include "/projects/robot/rccl/h/rci.h"
#include "/projects/robot/rccl/h/macros.h"
#include "RTC.h"

#define FOR_DRIVE_MODE  0       /* selectcompliance law                      */
#define ROTATE_MODE     1       /* select compliance law                     */
#define BACKDRIVE_MODE  2       /* selects compliance mode                  */
#define FOR_THRESHOLD   50.0    /* f/t units difference stop moving forward */
#define ROT_THRESHOLD   200.0   /* f/t units difference before terminate    */
#define POS_THRESHOLD   0.01    /* Displacement threshold                    */
#define BACK_THRESHOLD  -8.0    /* F/T units along dir vector to drive */
int     scnt;
float forceValues[536][6];

real            gain;           /* gain for the compliance motion          */
                                /* NOTE: must be initialized by calling    */
                                /* routine.                                */
short           initforce[6];   /* initial force before the move           */
                                /* should be biased out                    */
VECT            dir;            /* "unit" direction vector indicating      */
                                /* direction of movement with respect to   */
                                /* force torque sensor frame               */
real            maxdist;        /* desired linear displacement             */
int             evfnResult;     /* 0 - if no contact made                  */
                                /* 1 - if translational contact is made    */
                                /* 2 - if rotational contact is made       */
                                /*     and transl. contact was made        */

real            for_drive;      /* distance (mm) to move during sample     */
                                /* period when in FOR_DRIVE_MODE           */
                                /* NOTE: must be initialized by calling    */
                                /* routine.                    */

real            back_drive;     /* distance (mm) to move during sample     */
                                /* period when in BACK_DRIVE_MODE          */
                                /* NOTE: must be initialized by calling    */
                                /* routine.                    */

real            rot_drive;      /* distance (mm) to move during sample     */
                                /* period when in ROTATE_MODE         */
                                /* NOTE: must be initialized by calling    */
                                /* routine.                    */

int             mode;           /* Flag indicating whether in              */
                                /* FOR_DRIVE_MODE or BACK_DRIVE_MODE       */
float           rx, ry, rz;     /* temp var to simplify (n o a) vector calc*/
float           rotx;           /* components in tool frame to rotate    */
float           roty;          /* about to make planar contact between */
float           rotz;          /* plate and lug                         */

float           tdiff;          /* found using transformed forces      */
                               /*  about point of contact             */

float           transfv3, transfv4, transfv5;
                               /* transformed force readings to frame  */
                               /* defined by translation by xcontact,  */
                               /* ycontact, and zcontact.              */

float           xcontact;      /* locates contact point; used in force  */
float           ycontact;      /* transformations                      */
float           zcontact;      /* should be imported from planning file  */


find_rot_evfn(t)
        TRSF_PTR        t;
{
        int             j;      /* counter */
        float           fdiff;  /* force difference */
        real            pdiff;  /* distance between current position */
                                /* and desired terminal position     */


        fdiff = (how.forceValues[0] - initforce[0]) * dir.x +
                (how.forceValues[1] - initforce[1]) * dir.y +
                (how.forceValues[2] - initforce[2]) * dir.z;

/*store forces in 2-D array every sample period*/
        if (scnt > 536) {
        evfnResult = 5;
        rcl$terminateMotion = YES;
        return;
        }
else {


        for (j = 0; j<6; j++)
            forceValues[scnt][j] = how.forceValues[j];
}
    scnt++;

        switch (mode) {
        case FOR_DRIVE_MODE:

        if (FABS(fdiff) < FOR_THRESHOLD) {
            pdiff = maxdist - sqrt(
                    (t->p.x * t->p.x) +
                    (t->p.y * t->p.y) +
                    (t->p.z * t->p.z));

        if (FABS(pdiff) < POS_THRESHOLD) {
                                                        /* Motion completed */
                evfnResult = 0;                         /* without hitting */
                rcl$terminateMotion = YES;              /* anything */
                return;
                }
                if (FABS(pdiff) < for_drive) {          /* Very close to  */
                  for_drive = FABS(pdiff);              /* terminal position */
                }                                       /* so must adjust */
                                                        /* positional step */
                                                        /* size */


                gain = for_drive / FOR_THRESHOLD;
                t->p.x += (for_drive + fdiff * gain) * dir.x;
                t->p.y += (for_drive + fdiff * gain) * dir.y;
                t->p.z += (for_drive + fdiff * gain) * dir.z;

                } else {
                        evfnResult = 1;
                        rcl$terminateMotion = YES;
                        return;
                }
                break;

        case BACKDRIVE_MODE:    /*to relieve forces but still keep contact*/

                if (FABS(fdiff) < BACK_THRESHOLD) {
                    evfnResult = 2;
                    rcl$terminateMotion = YES;
                    return;
                } else {
                        t->p.x -= back_drive * dir.x;   /*backdrive in dir*/
                        t->p.y -= back_drive * dir.y;   /*of motion       */
                        t->p.z -= back_drive * dir.z;
                }
                break;

        case ROTATE_MODE:

/* Transforms forces to a frame at point of contact with same */
/* orientation as tool frame                                  */
/* Torque scale factor for this tool is 4                 */
/* Should subtract out transformed bias                   */
                if (FABS(tdiff)<ROT_THRESHOLD){
                        transfv3 = how.forceValues[3]*4 -
                                   ycontact * how.forceValues[2]-
                                   zcontact * how.forceValues[1];
                        transfv4 = how.forceValues[4]*4 -
                                   zcontact * how.forceValues[0]+
                                   xcontact * how.forceValues[2];
                        transfv5 = how.forceValues[5]*4 +
                                   xcontact * how.forceValues[1] +
                                   ycontact * how.forceValues[0];

/* torque about new rotation vector*/
                        tdiff = transfv3 * rotx +
                                transfv4 * roty +
                                transfv5 * rotz;



/*                      tdiff = (how.forceValues[3] - initforce[3]) * rotx +
                                (how.forceValues[4] - initforce[4]) * roty +
                                (how.forceValues[5] - initforce[5]) * rotz;
*/
                        gain = rot_drive/ROT_THRESHOLD;
                        rx += (rot_drive + tdiff*gain)*rotx;
                        ry += (rot_drive + tdiff*gain)*roty;
                        rz += (rot_drive + tdiff*gain)*rotz;

                        t->a.x = ry;    /* Assumes differential rotations*/
                        t->a.y = -rx;   /* 2nd order terms discarded    */
                        t->o.x = -rz;   /* to save computation time     */
                        t->o.z = rx;
                        t->n.y = rz;
                        t->n.z = -ry;

                }
                else {
                        evfnResult = 2; /*has made planar contact*/
                        rcl$terminateMotion = YES;
                        return;
                    }
                    break;
/* Add RotateBackDrive here*/

        }}

/* find_rot_evfn.c */
```