NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

Canada

ACTIVE FORCE-CONTROLLED PART ASSEMBLING FOR A ROBOTIC ASSEMBLY CELL

by

Badreddine Karoui

A Thesis Submitted to the

School of Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Sciences

Ottawa-Carleton Institute for Electrical Engineering

Departement of Electrical Engineering

Faculty of Engineering

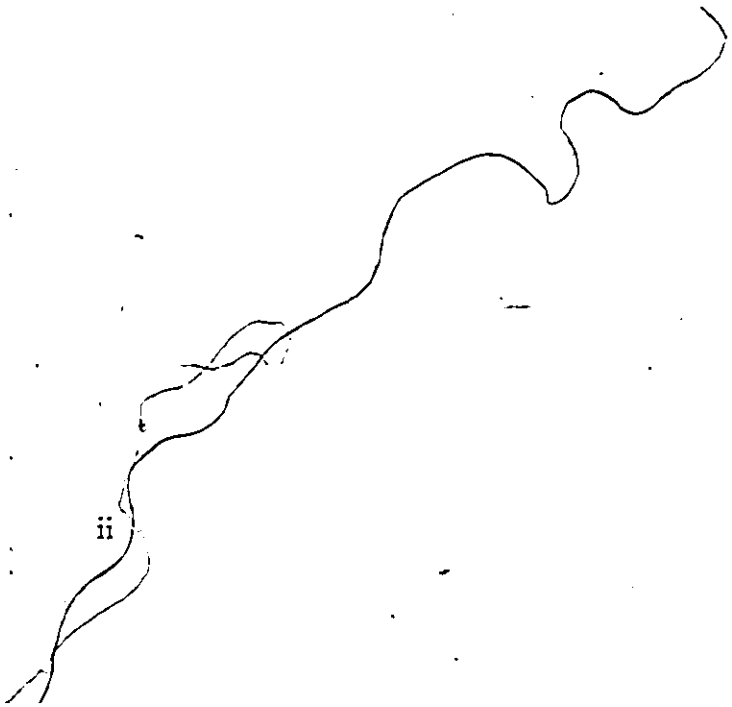
University of Ottawa

May 1988

ISBN 0-315-46786-X

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

The University of Ottawa requires the signatures of all persons using or photocopying this thesis. Please sign below, and give your address and the date.



Abstract

Assembling is a complex operation which involves a large portion of the production personnel. As a result, assembly is considered as one of the major areas of robotics application. Robotic assembly requires 4 major steps: (1) part acquisition, (2) part handling, (3) part positioning, and (4) part mating. This work addresses the issue concerning the last two steps that is part positioning and mating. These two steps include the placement of parts into their assembled position and then the joining of the parts together. During the mating phase, a small error in relative position or angular orientation produces very large forces and may damage the part or the robot arm. An active compliance provide the the solution for this problem.

This thesis presents a sensory-based control algorithm for part positioning and part mating implemented as a distinct functional unit. This functional unit is designed to be integrated within the structured framework of a complex assembly cell supporting a task-level programming language

Acknowledgements

Grateful thanks to Dr. Emil M. Petriu for his advice, guidance, and very constructive criticism during the course of my research. I would like also to thank Dr. Petriu for being a good friend and I hope that we will remain friends.

A special thanks to my friends in Canada and in Tunisia for their moral support. I also wish to express my appreciation to a beautiful friend, Christine Marceau, for her patient and understanding.

I gratefully acknowledge the Tunisian Government for providing me with the financial support throughout my study in Canada.

My deepest appreciation go to my family for their moral support and most of all for their love. I am specially grateful to my loving mother, Khadija, for all the care and love she is giving me.

Contents

List of Figures

Chapter 1

Introduction

Robotics has gone through a fast transition from theory to application. The use of the robots on the factory floor has lead to an increase in productivity. The main feature that made the robots attractive is their versatility, in conjunction with different sensors and end-effectors, the robot is capable of performing different tasks which make it adaptable to the variations in the production line.

Today's industrial robot's major application areas are: welding, painting, machine loading, and assembly which remains a challenge for industrial robotics. The largest number of robots is used in arc and spot welding. This application has a hot and hazardous environment and it is a very repetitive job. The robot will reproduce programmed tasks independent of the environmental constraints such as smoke and temperature. The use of the

robot in welding leads to an increase in productivity and quality. The positional accuracy and repeatability obtained by the robot is much better than that obtained by a human welder.

The hostile and undesirable environment of the painting in industry made this task an ideal application for the robot. Most of the robots used in this application are programmed by teaching. A skilled worker performs the motions of painting while the robot's microprocessor stores the motions through a teaching device. On playback the robot will duplicate the worker's motions.

Robots are also used with processing machines to unload the finished parts and to load the unfinished ones. In this way the robot eliminates the need to invest in part transfer equipment. In this application the robot is used in two basic configurations: (1) The robot serves a single machine or (2) it serves several machines. These machines must be of NC or CNC type since a communication between the robot and the machine must exist.

Building high-quality products at low cost has become the paramount business good of today's manufacturers. Assembly accounts for more than 50% of total manufacturing time. Based perhaps on this fact, most experts agree that the biggest potential for robots is in assembly which is described as the *sleeping giant of robotics.*

Three methods are currently used to assemble parts:

1. Manual assembly.

2. Special-purpose machines.

3. Robotics assembly.

More than 90% of consumer goods are still assembled by people because these goods have a low volume of production or a design life too short to justify investment in a special purpose machine. People use their skills to carry out the fine assembly operations. They are able to adapt to changes in product design and changes in the environment. However, people get tired and a repetitive assembly task is a boring job. People are also not able to perform the same task in the same way many times.

When the product volume is high, assembly of parts are carried out using special-purpose machines. The major drawbacks of these machines is their inflexibility with respect to changes in product design, they are very expensive and usually handle only one set of parts. Special-purpose machines are limited to assemble products that are produced in millions for many years: such a volume of production represents a small percentage of all the goods produced.

It has been shown that assembly using programmable robots can be very profitable for mid-volume production levels. A robot can be taught to perform a new assembly task or several tasks in sequence. Because of robot programmability, simple workpiece feeders and fixtures may be used in a robotized assembly system.

Nowadays, the number of robots dedicated to assembly tasks is still small compared to other applications. In Japan, Europe and USA, robots have been assembling electronic components, typewriter keyboards, appliance parts and automobile bodies. As the interest increases many companies start designing and manufacturing assembly oriented robots. In 1975 the Italian company Olivetti introduced the Sigma robots which have been used for electronic component insertion and typewriter keyboard assembly. Many other robots followed, some of them have names that underline their assembly features. In 1978 the PUMA (Programmable Universal Machine for Assembly) was introduced by Unimation.In Japan, in 1979 the SCARA robot (Selective Compliance Assembly Robot Arm) was introduced.

While using robots for assembly seems very profitable, many difficulties are still to be overcome. Two main problems can be distinguished. The first problem is that the existing manufacturing framework is not suitable for robotics application. To solve this problem new product design guidelines have been introduced. This new guideline is to make the product adaptable to assembly by robots. The following is a summery of these guidelines [1]:

- Reduce the number of parts which must be joined together.
- Replace screwed connections by locking connections.
- Add guiding slopes and chamfers to holes and cylindrical components.
- Design part surfaces that can be gripped by robots.

- Eliminate components which are difficult to be gripped or be fed.
- Design symmetric parts which facilitate the feeding and orientation in the assembly.
- Special consideration should be paid to the optical characteristics of components (color, surface finish, light reflection, etc.)
- Reduce the complexity of final assembly by defining new subassembly groups.

The second problem is related to the dimensional tolerances of the components that can make the mating of parts very difficult. While the positional tolerances of the parts to be mated are very high, in the range of one hundredth of a mm, the accuracy of current manipulators are only in the range of one tenth of a mm. This fact makes the assembly of high tolerance parts theoretically impossible. The low accuracy of robot arms leads to errors in relative position and angular orientation between the parts to be assembled. Due to these errors very large contact forces occur and prevent the completion of operation and may cause damage to the parts or the robot arm. It has been shown that robotic mechanical assembly requires the control of both force and position in order to guarantee the success of the task. The force information can be obtained by a force sensor and it can be used to make small changes in the path of the end-effector when the parts are being assembled.

Robotics, which was once defined as the "intelligent connection of

perception to action", requires a system approach. The development of a robotics based assembly system has to be done in a structured way, allowing for a better integration of all components: robots, sensors, feeders, conveyors, etc. One of the goals of the intelligent robotic systems is the task-level programmability.

An assembly task is viewed as a successive convergence of unoriented objects with increasingly precise relative positioning until mating, subject to contact forces, can take place. The task specification is decomposed into two sub-tasks: object fetching, and object assembly. This thesis discusses the problems related to object assembly sub-task that is the *part positioning* and *part mating*. With the exception of chapter2, the rest of the chapters are dedicated to the position and force control of the robot arm during the object assembly sub-task.

The objective of this thesis is the development of the force controlled function to be integrated within the structured framework of an assembly cell conceived to support a *task-level programming language*.

In the case of the task-level programming, the assembly tasks are specified in terms of operations on objects rather than on the movements of the robot. Chapter 2 discusses the architecture of the robotic assembly cell conceived as a test-bed for such a task-level programming concept. Interfacing problems between system elements such as manipulator arm, vision camera, force sensor, conveyor, and other external devices are mentioned. It also includes a brief introduction of the proposed task-level program-

ming language to be implemented. The remaining chapters deal with the development of the position and force control for the assembling unit of the assembly cell.

Chapter 2

Robotic-Assembly Architecture

2.1 Introduction

In its early stage, robotics was mainly concerned with motion. From the application experience, the robotics community came to realize that the key problem was actually the integration of a broader spectrum of capabilities: manipulation, sensing, and intelligence [2].

One of the most desired feature in an assembly station is flexibility which allows manufacturers to convert an assembly line from one product to another with no major changes. Based on the robotic handling of

the objects, computer control, and multi-sensor feedback, flexible assembly cells are currently being developed. The use of a robot to handle the parts to be assembled represent the main factor which gives the assembly cell flexibility. The existence of the computer control adds the programmability feature to the assembly station. The multi-sensor feedback brings to the assembly station the needed feature of adaptability with the environment. The integration of manipulation and sensing capabilities into working systems incorporates a hierarchical framework of control, architecture, programming and knowledge representation [3].

The architecture of the assembly cell developed in our laboratory is conceived as a structured frame work for a task-level programmable robotic assembly system. Unlike robot-oriented programming, task-level programming specifies the assembly task in terms of operations on objects being manipulated rather than by the robot motions. In this way the task of programming is simplified.

This chapter is conceived as a general chapter to introduce the overall framework of a task-level programmable robotic system within which the position and force controlled assembly is developed. The functional requirement and system integration are stressed for the design of an assembly system which emphasize throughput and flexibility.

2.2 System Structure

The structure of the assembly station is shown in figure 2.1; a photograph of the set-up of this assembly station is shown in figure 2.2. The components of the assembly station are as follows:

- a robot arm with gripper;
- a conveyor;
- position sensor;
- a force/torque sensor;
- an assembly fixture;
- a vision system.

This system is able to perform all the steps needed by an assembly task: acquisition, handling, position, and parts mating.

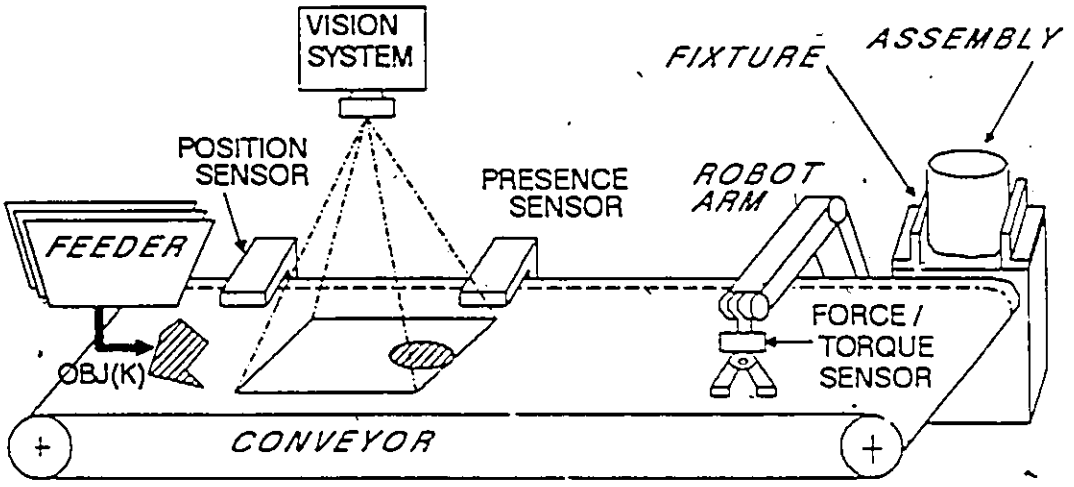


Figure 2.1: The layout of the assembly station

Parts can be presented by mechanical devices that result in a high positional accuracy. Such devices are usually used with only one type of parts which have the disadvantage of restricting the flexibility of the assembly cell. The use of sensing rather than fixed mechanism to resolve uncertainties is a key point in flexible assembly systems. In conjunction with a conveyor, a vision system is used to provide a flexible part presentation system. During the acquisition phase, an object, previously specified, is fed onto the conveyor belt. A presence sensor detects the object and triggers both the vision system and the position sensor. The vision system and the position sensor provide the absolute position of the object. The positional information is then used by the robot to track and grasp the object. This situation is illustrated in figure 2.3 where the progressive image transmission technique can be seen [4]. The presence sensor decreases the

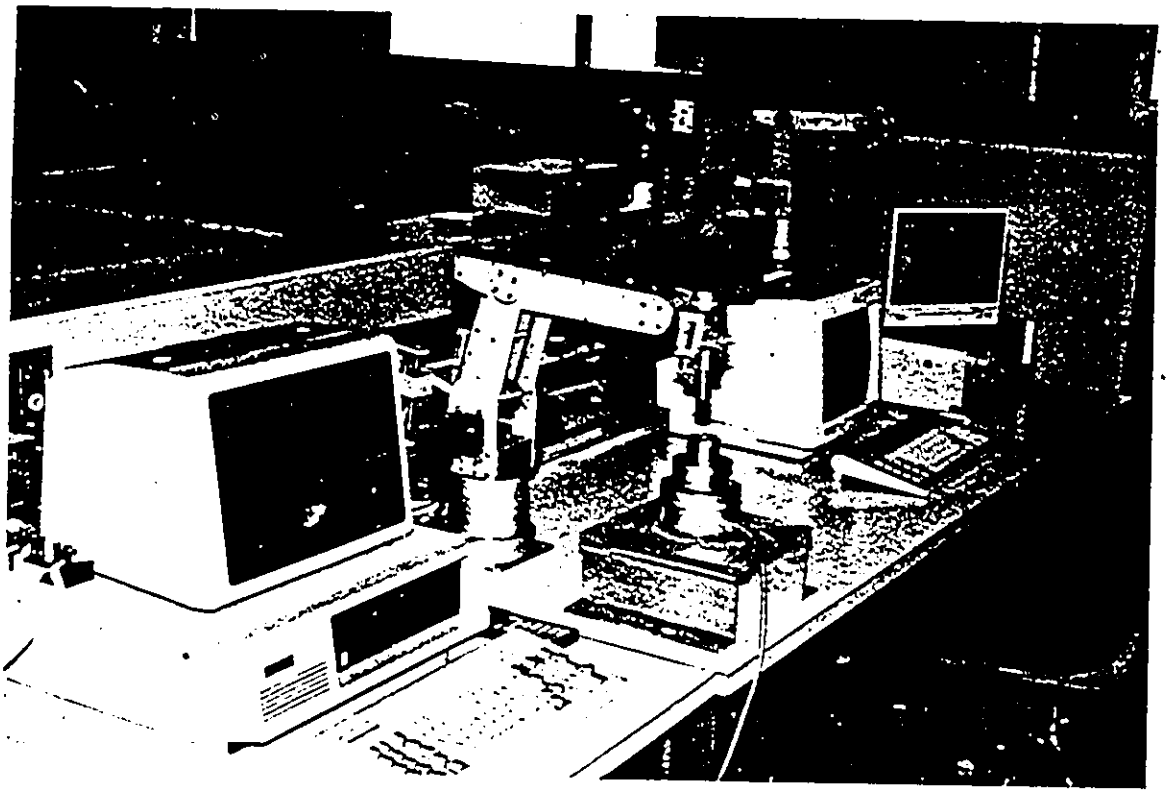


Figure 2.2: The assembly station

object position entropy, then the vision system brings this entropy to an acceptable level for the robot. The progressive vision technique has the advantage of initiating movement of the robot arm earlier than if a normal vision techniques would be used. Once the object is grasped the robot arm executes the orientation and positioning phase which insure that the current part is properly positioned relative to other parts prior to mating. The last phase is the part mating. Measurement of the interaction forces and torques is the most reliable source of information to guide the robot during this phase. This measurement is done through the force/torque sensor.

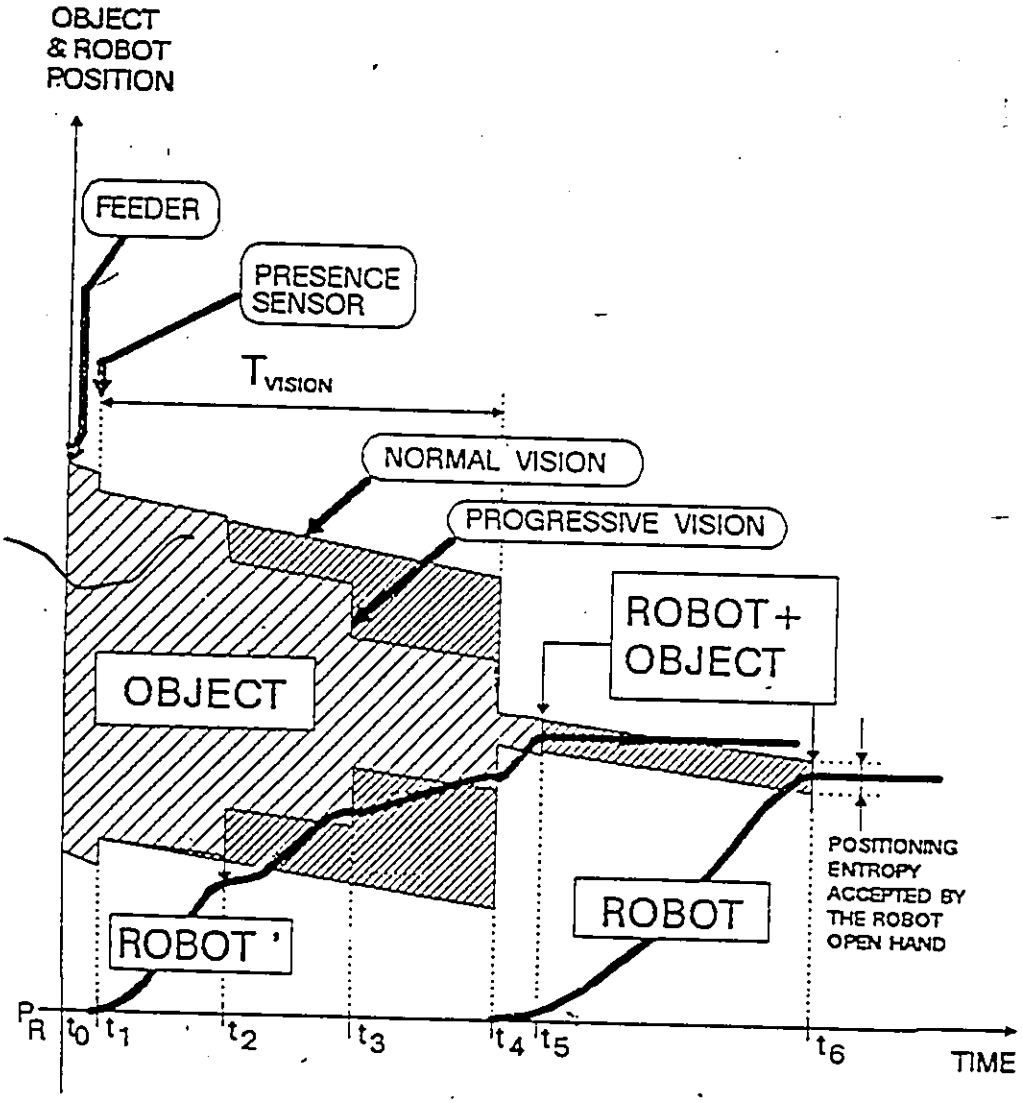


Figure 2.3: Robot tracking using progressive image representation

2.3 Task-level programming

The principal advantage of the robots over traditional automation is versatility. This key characteristic can be fully exploited only if the robot can be programmed easily. Robot programming play a crucial role in robotics development. The earliest (and still the most widespread) method of robot programming is known as *guiding* (teaching by showing). Programming by guiding cannot make use of sensors and consequently is inappropriate for assembly applications. In order to overcome this drawback, the *robot-level* programming enable the use of the sensory information allowing the robot to cope with a greater degrees of uncertainty in the position of external objects. The main drawback of this programming technique is that it requires a high qualified robot programer. The programmer should be an expert in computer programming and in the design of sensor-based motion strategies [5].

A more recent approach to robot programming is the *task-level* programming which is completely robot-independent. Task-level programming specifies tasks by their positional goals rather than by the motions the robot has to perform to achieve this goals. In contrast to guiding and robot level programming, task-level programming is still in its research stage.

2.3.1 Task specification

Task-level programming allows for the task specification to be in terms of operations on the objects. The assembly is viewed as a sequence of simple assembly operations on objects. This sequence of operations performed on a physical object may be regarded as an assembly algorithm. The assembly algorithm is a procedure for the execution of a given task, but it is not specific to implementations or any system. A task planner transforms the task-level specification into robot-level specification. The task planner architecture described further is similar to that described by Gonzalez [6]. Figure 2.4 highlights the organization of the task planner. The task specification is decomposed into two sub-tasks, object fetching and object assembly. Sub-task planners generate programs for the implementation of the sequence of the assembling operation to which objects are submitted:

1. Object presentation (involving the conveyor, presence and position sensors as well as vision).
2. Object tracking and grasping (involving the robot arm).
3. Object orientation and relative positioning (involving the robot arm).
4. Object mating (involving the robot arm and the force/torque sensor).

Task-level programming systems require complete geometric modes of the environment and objects.

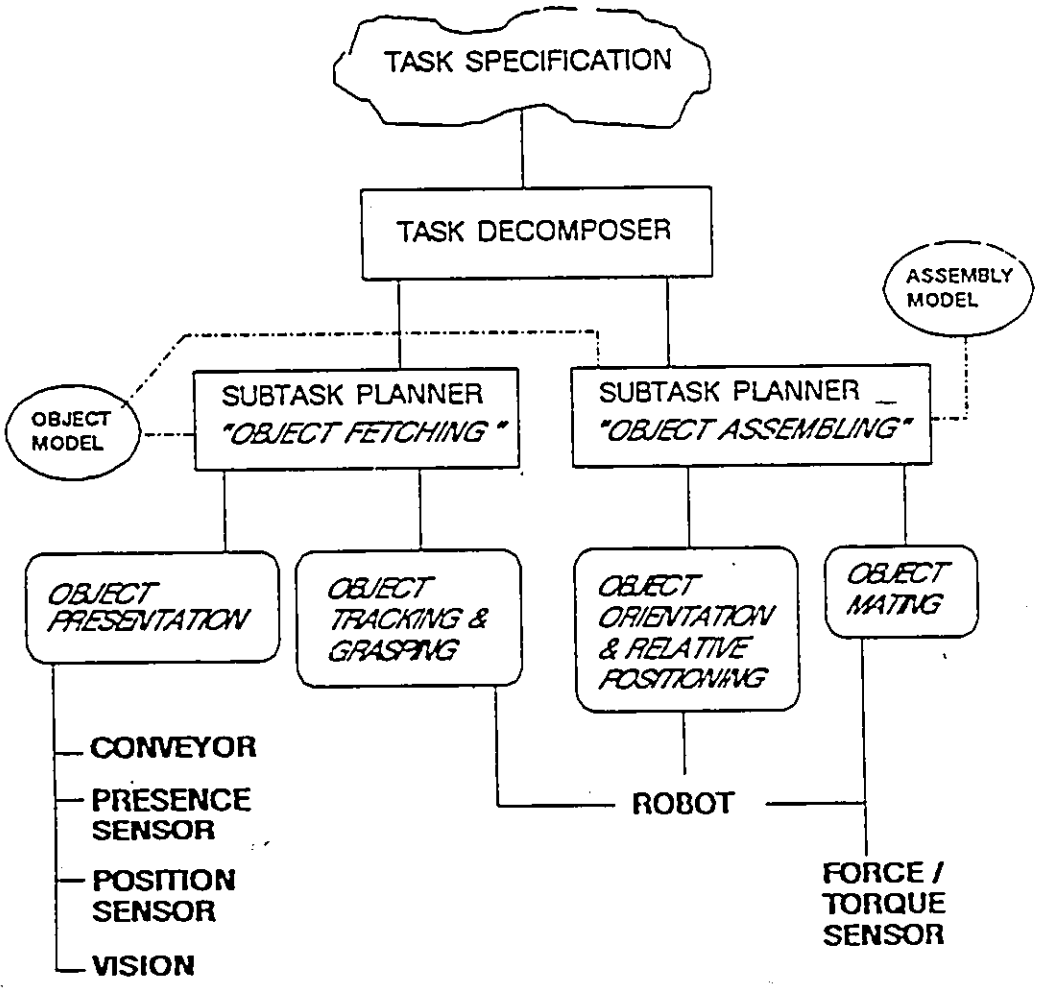


Figure 2.4: Task planner

2.3.2 Instruction Format

The task level language has not yet been fully specified. What follows is just the syntax of the instruction defining the assembly tasks [7]:

<OPERVERB> <OBJNAME> <CONTCTREL> [<MODIF>] [<CONSTR>]

Where:

<OPERVERB> ::= SIMP_PEG_IN_HOLE | PUSH_TWIST | MULT_PEG_IN_HOLE | PEG_RETAIN | SCREW | FORCE_FIT | RMV_LOC_PIN | FLIP_OVER | PROV_TEMP_SUP | CRIP_SHT | RMV_TEMP_SUP | WELD_SOLD;

<OBJNAME> ::= list of legal object names as predefined in the object model in the task specification e.g., PEG1, PEG3, BOLT, WASHER10, BLOCK8;

<CONTCTREL> ::= <OBJSIDE><RELTYPE><ASSEMSIDE>;

<MODIF> ::= GUARDED | FREE_MOVE | etc.;

<CONSTR> ::= TORQTHR .EQ. value | FORCETHR .EQ. value | etc.;

<OBJSIDE> ::= list of object sides as predefined in the object model in the task specification e.g., PEG3_SIDE1;

<ASSEMSIDE> ::= list of the object sides acceptable to the object model where the object is already part of the assembly e.g., BLOCK8_SIDE4;

<RELTYPE> ::= AGAINST | FIT | COPLANAR;

2.4 Architecture of the Assembling Processor

The architecture of our robotic assembly system is shown in Figure 2.5. This model identifies the two functional units, the object presentation system and the assembling processor, corresponding to the two object assembly sub-tasks referred to in Figure 2.4. Also depicted in Figure 2.5 are the interface signals for communication between the functional units. It should be mentioned that the "task decomposer" level of the task planner is actually executed by the the assembling processor which is in "master-slave" relationship with the object presentation system. The assembling processor consists of a robot arm with controller, a fixture, a force/torque sensor, and a personal computer. Both the force/torque sensor and the robot controller are connected to the computer via an RS-232 interface. The fixture is a rigid table on top of which the force/torque sensor is fixed. A detail discussion of the force/torque sensor is presented in Section 3.2.3. The next section is dedicated to the description of the robot system.

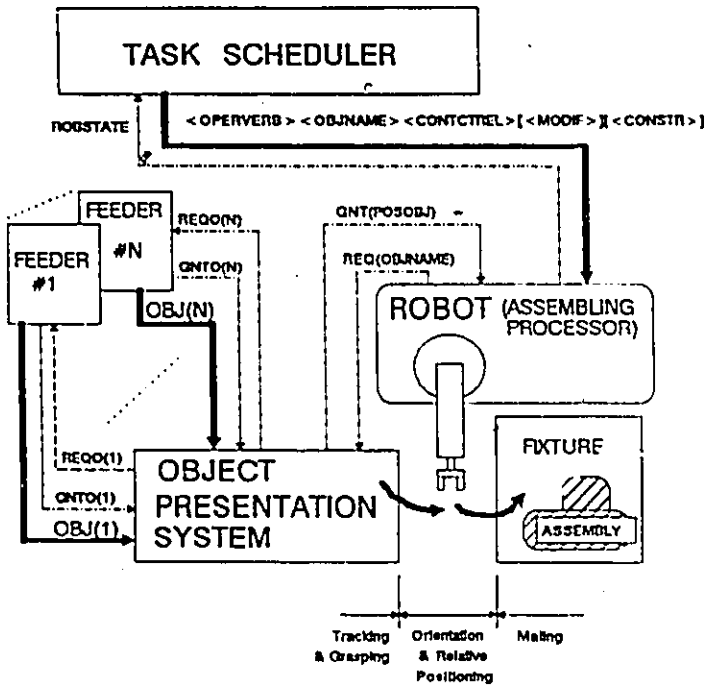


Figure 2.5: Architecture of the assembly cell

The Robot

Robots used in assembly perform the following tree major functions: positioning of the arm and its end-effector, grasping of the part, and force/torque application to part or assembly. The assembling processor makes use of the first and third one of these functions.

We are using an SIR-1 robot system consisting of a robot arm, a microprocessor-based controller and a hand-held teach pendant. It can operate in either teach mode or from a host computer. The teach mode has no practical use in this work and is not discussed. (For more information related to the teach mode the reader is refered to the "User manual of the SIR-1"). A command set for the computer link is provided [4]. This

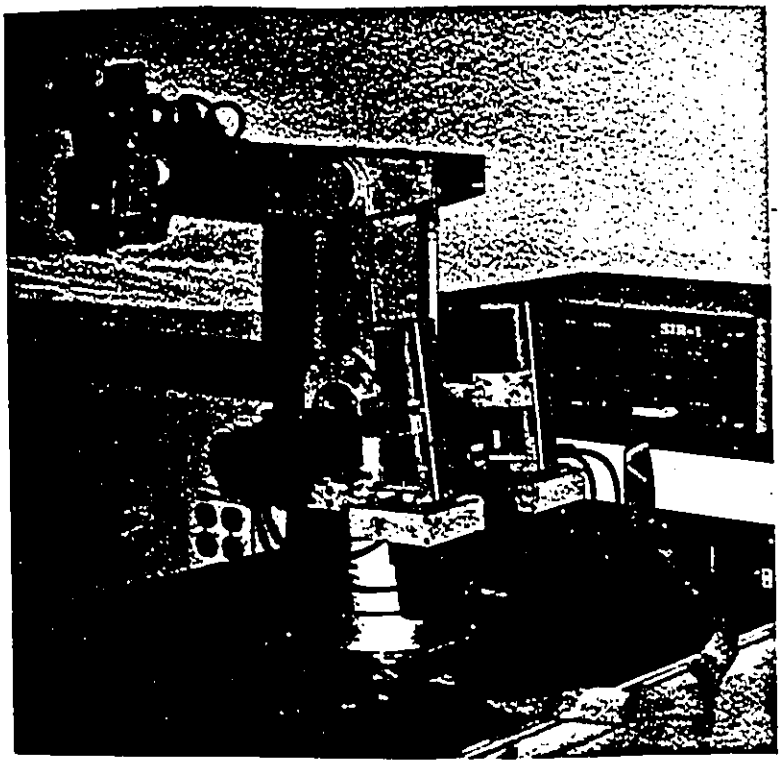


Figure 2.6: SIR-1 robot system

command set is to be implemented in a host program which works as an I/O interface between the computer and the robot system. The command set can be found in Appendix A.

SIR-1 robot arm

SIR-1 is an articulated robot arm with five degrees of freedom. Two actuators are coupled to links 2 an 3 via four-bar linkages. Joints 4 and 5 are operated through a chain drive by two other actuators.

The drive system of the robot arm and its gripper consists of six DC servo-gearmotors which are equipped with incremental shaft encoders. This

arm has 11 software end-of-travel limits and 6 limit switches placed after the software limit. The pay-load of the arm is 2 kg. The manufacturer claims that a repeatability of ± 0.6 mm can be obtained. The maximum speed that can be obtained, at the finger tip, is 460 mm/sec.

SIR-1 controller

A microprocessor-based controller interprets the commands sent by the host computer or given through the teach- pendant. It controls simultaneously 8 DC servo-motors : 6 on the robot arm and 2 extra ones that can be used on an accessory. The controller is equipped with I/O ports (four input bits and four output bits) and provides the physical connection for the computer link. This connection is obtained through a standard RS-232 C serial interface port.

Chapter 3

Position and Force Control for Assembly Operation

3.1 Introduction

Most of robot control systems make robots behave as position sources. Many robot tasks such as spot welding and painting can be specified as a sequence of positions. Uncertainty and error in the positions of objects are inherent aspects of any robot manipulation task. Furthermore the effect of commanded motions is not completely predictable. Sensory devices provide information which will reduce the uncertainty of parts position. Contact with objects whose position is not exactly known is a crucial part of any manipulation task. Many assembly tasks can be specified in an

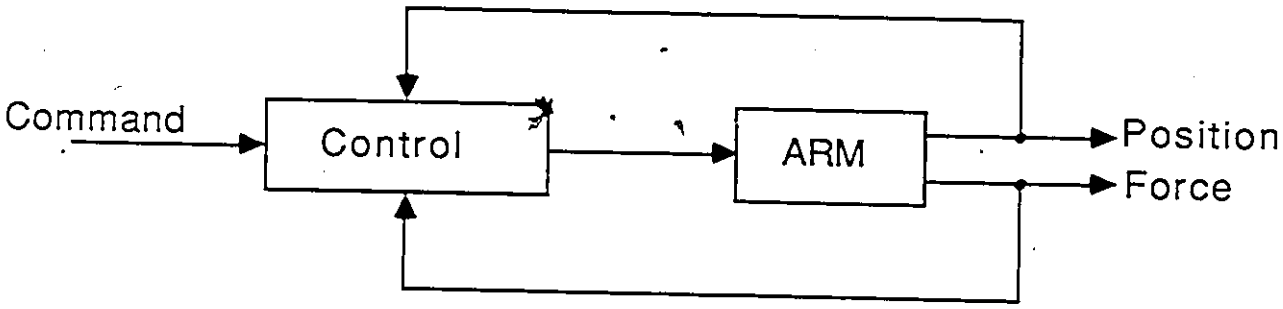


Figure 3.1: General position/force control loop.

efficient way by a desired relationship between forces and displacements. This type of specification is less sensitive to uncertainty and errors than a pre-determined position trajectory [10].

Robot force control has been investigated for several years, many sensing methods and control strategies have been proposed. While the main goal of most of the published work has been the analysis of force control strategies, one should note that a force control must always be used in conjunction with a position control. Most often, the forces are controlled along certain Cartesian degrees of freedom while the position is controlled along the remaining degrees of freedom. Figure 3.1 shows the general position/force control loop.

The situation illustrated in Figure 3.1 uses a force sensor in the force feedback loop. This type of control is called *active force control* or *active compliance.*

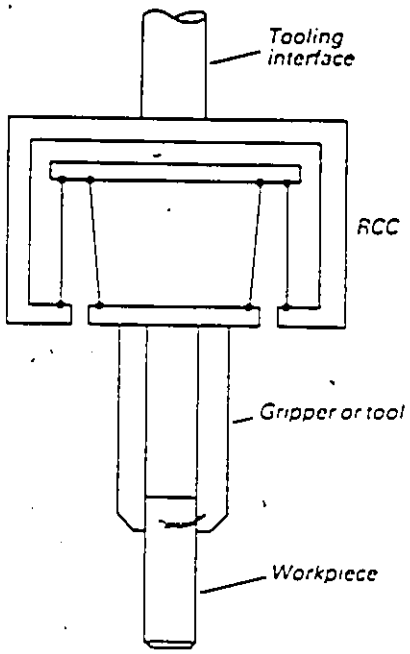


Figure 3.2: The remote center compliance wrist (RCC) [4].

Another approach of compliance is to provide the hand with an elastic structure. The hand is then able to deform in such a way that the external forces are minimized. This control method is called *passive force control* or *passive compliance*. The remote center compliance wrist (RCC), shown in Figure 3.2, developed at Draper labs [4], allows positional and angular misalignments between parts to be accommodated in such a passive way.

Our approach through this is work to use the active force control. This control scheme is further discussed in Section 3.3.

3.2 Force Sensing Techniques

Force sensing is fundamental for the implementation of any active compliance control mechanism. The existing force sensing techniques can be classified into two major categories.

1. Joint frame sensing .
2. Cartesian frame sensing.

The Cartesian techniques can be further divided into two categories according to the placement relative to the robot arm of the sensing devices.

i. Sensing at the wrist.

ii. The sensor is mounted on the fixture or support platform of the object to be handled.

3.2.1 Joint Frame Sensing

This techniques is perhaps the oldest. It has been used in master-slave manipulator systems allowing the operator to feel the type of forces that the slave arm is experiencing. This technique consists of measuring the joint torques of the manipulator by either monitoring the motor's current (for electrically powered arms) or the back pressures (for hydraulically powered

arms), or by mounting a special joint torque sensor.
The equivalent Cartesian force and torque can be obtained using :

$$F = (J^T(\Theta))^{-1}\tau \tag{3.1}$$

Where, F is a Cartesian force-torque vector acting at the end-effector; τ is a joint torques vector and $J^T(\Theta)^{-1}$ is the inverse of the Jacobian transpose. This conversion implies a major disadvantage: it involves the inversion of a matrix (usually 6x6) which is time consuming. A further disadvantage is that the friction and the joint damping are unpredictable, this prevents measuring precise forces at the end-effector.

3.2.2 Cartesian Frame Force Sensing

Measuring force-torque at or close to the end-effector eliminates the disadvantage originating from converting the joints torques. This can be accomplished by mounting a force-torque sensor at either the wrist or the part to be handled.

i. Sensing at the wrist

This technique consists of mounting a force sensing device below the last joint and close to the end-effector. This has the advantage of eliminating the friction and the dynamic effects from the measurement of the external forces. However, the inertia and loading added by the sensor has

to be taken into account and compensation for the inertia and gravity of the hand is needed.

Force sensing devices vary in complexity from two degrees of freedom to six degrees of freedom. In section 3.2.3, more discussion on these devices is presented.

ii. Force Pedestal

The force sensor is mounted on the part to be handled eliminating the problem of the extra loading as well as the friction at the joint and the links dynamics. However, since in an automated assembly system thousands of parts have to be handled, it would be financially disastereous to mount a force sensor on each part. One acceptable solution is to mount the sensor on the fixture on which the parts are handled. This technique is adopted through this work for the following reasons:

- The robot arm used for this work has a pay-load of 2kg, the force sensor used has a weight of approximately 2kg. Mounting the sensor at the wrist would make the robot work at its maximum capacity.

- The external forces are measured directly and no compensation is needed.

3.2.3 Six-Axis Force-Torque Sensor

Many force-torque sensors are available today. They are using different techniques to detect forces. Most of the force-torque sensors are strain gauge-based. Victor Scheinman [11] has designed a force sensor with 16 strain gauges (Fig 2.3). Based on this idea, Draper laboratories has developed a pedestal and wrist force sensor [12]. Metal foil strain gauges are used as the transducing elements. The design is basically the same for both pedestal and wrist sensors.
A compact six-degree-of-freedom sensor based on a deformable elastic structure and a set of inductive displacement transducers was developed at the institute of micro-engineering of EPF-L [13].

The sensor used through this work is the **Astek FS6-120A-100**. It measures forces and torques along three orthogonal axes. It is a foil strain-gauge-based sensor. It has an internal microprocessor which converts raw data into forces and moments along the axes. Both analog and digital electronics are inside the sensor housing (Fig. 3.4). Several software features are available for the sensor (see appendix B) [14]. Forces up to 200N are measured with a resolution of 0.1N. Torques up to 4Nm are measured with 0.002Nm resolution. The data rate is programmable from 1.8 to 240Hz.

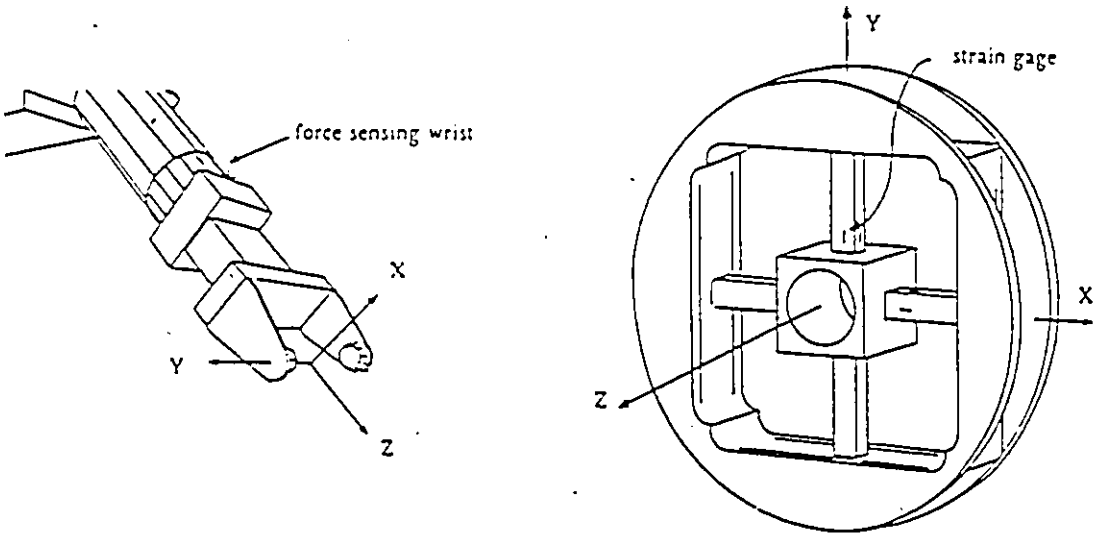


Figure 3.3: Scheinman's force sensing wrist [11].

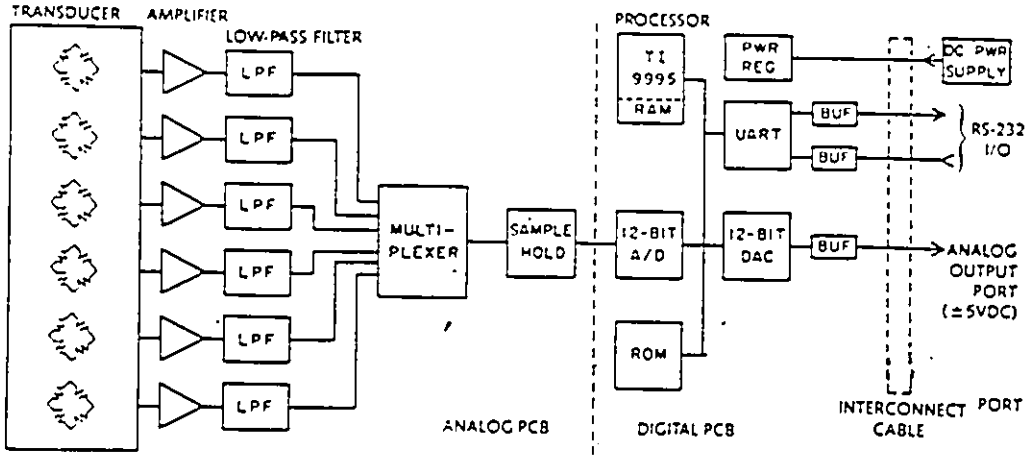


Figure 3.4: Electronics hardware of the Astek torque/force sensor [14].

3.3 Assembly Operations

Robot assembly has been under investigation for several years. Assembly tasks such as inserting electronic components on circuit boards, placing armatures, inserting valves in cylinders are considered a promising application for an industrial robot. At Drake laboratories [15],a study to catalogue the various kinds of assembly tasks needed to assemble many typical products gave the result shown in Figure 3.5. Most of the products can be assembled with various combinations of these 12 basic operations. The peg-in-hole operation is found to be the most frequent assembly operation and outnumbers the rest (Fig. 3.6).

Peg-In-Hole Operation

In any assembling there are two parts involved: the first is a fixed part with a hole, and the second part is the peg. Inserting a peg into a round hole is the single most frequently studied task in robotics assembly research. This task is basically a positioning problem. Observation of the operation [15] revealed that the peg goes through three main stages before the completion of the insertion. First the peg crosses the chamfer, it then touches one side of the interior (one point contact), finally it touches the opposite side of the hole (two point contact) see Figure 3.7.

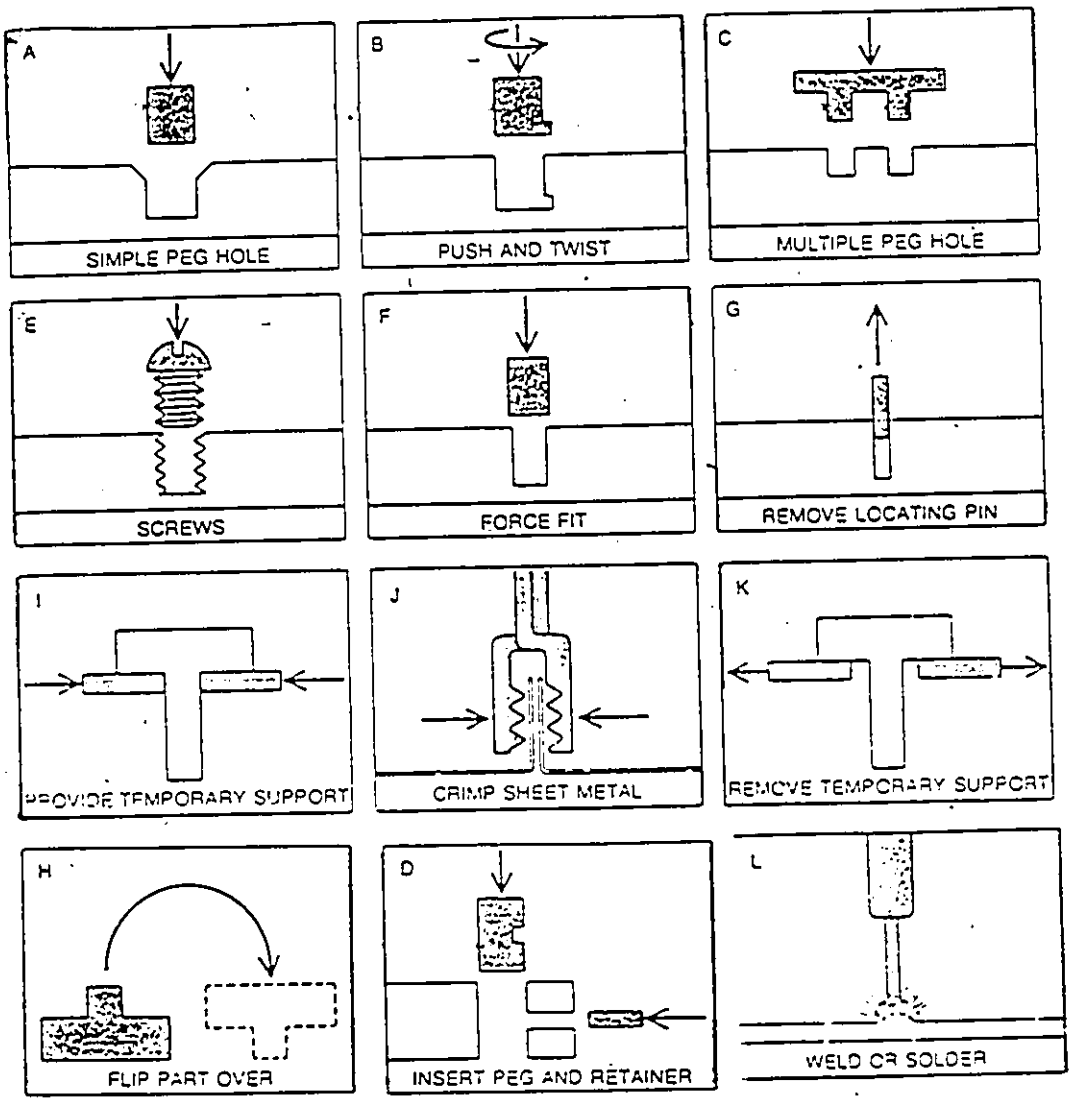


Figure 3.5: Typical manufacturing operations [15].

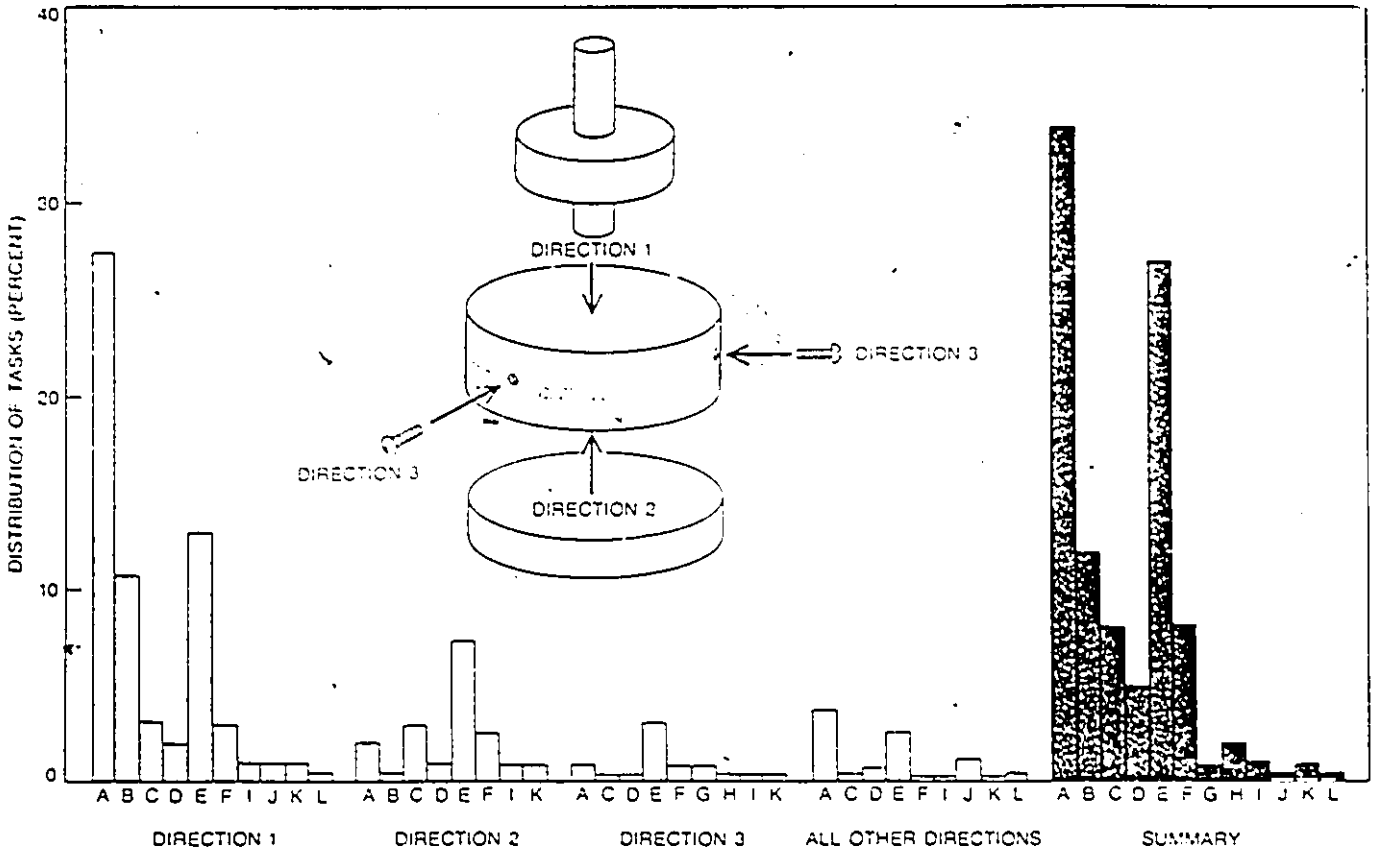


Figure 3.6: Frequency of the different operations [15].

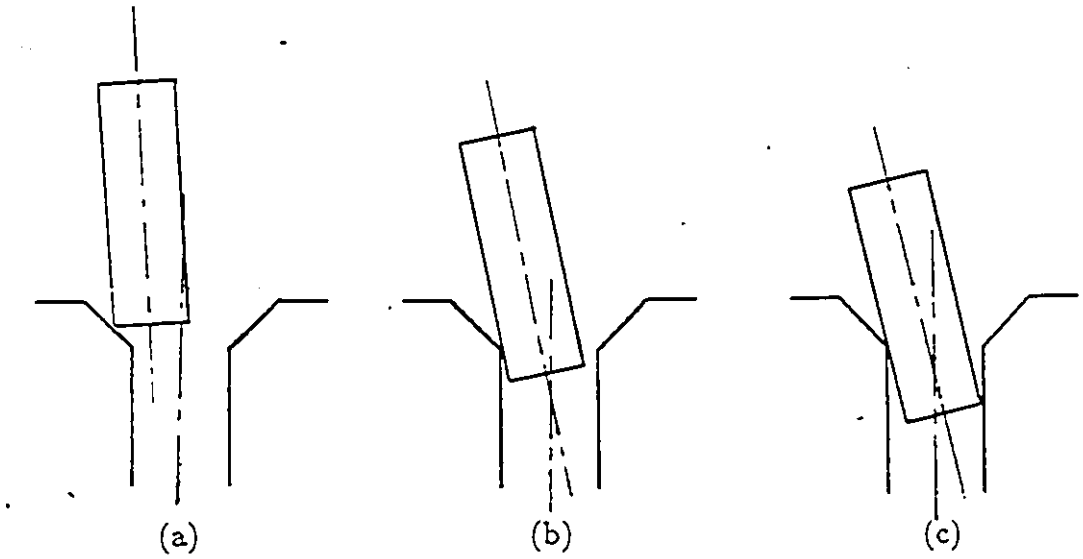


Figure 3.7: Peg-in-hole operation (a) chamfer crossing. (b) One point contact. (c)Two point contact.

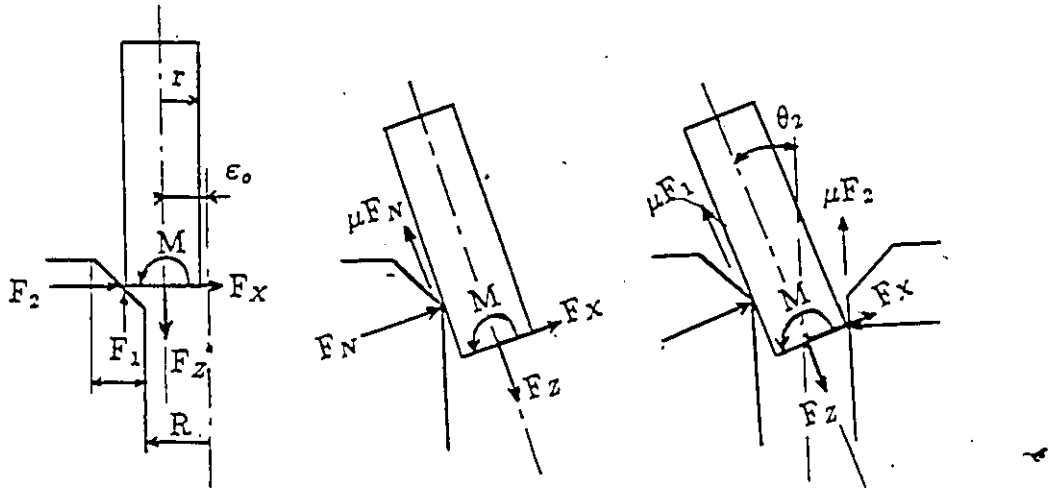


Figure 3.8: Forces acting on the peg [16]

The clearances and the shape of mating parts determine the difficulty of this task. D.E.Whitney studied the problem of high-tolerance peg-in-hole insertion [16]. This study resulted in the design of the RCC [17]. The results of the study can have a general application. Whitney first derived the forces acting on the peg in the different stages. Figure 3.8 shows the forces acting on the peg. Two problems may occur during the mating operation: When the forces applied are in a wrong direction or magnitude and prevent the insertion to proceed, the peg is said to be *jammed*. Another situation may occur when the peg will not move under any applied force, the peg is said to be *wedged*.

For successful completion of the insertion Whitney has derived sufficient conditions to avoid both problems. These conditions are summarized

below [17]:

1. To avoid wedging, two point contact must occur at a value of θ_2 such that:

$$\theta_2 < \frac{C}{\mu} \tag{3.2}$$

2. To avoid jamming, we must maintain

$$\frac{M}{rFz} + \mu(1+\lambda)\frac{Fx}{Fz} < \lambda \qquad with \qquad \lambda = \frac{l}{2r\mu} \tag{3.3}$$

and

$$\frac{Fx}{Fz} < \frac{1}{\mu} \tag{3.4}$$

where

$C = \frac{R-r}{R}$: clearance ratio.

μ : coefficient of friction.

3.4 Previous Investigation In Force Control

The robotics phenomenon is not the product of the 20th century. Robot-like devices existed as early as 500 B.C [18]. Until the 1950s these were mechanical devices with no feedback from the environment. In the early 1950s, Goertz invented an electric-servo master-slave manipulator

with force reflection for radioactive labs. The operator feels the forces experienced by the slave joint through the master joint. In the 1960s, Mann developed, for the amputee, a force feedback powered artificial elbow. The joint motor is driven by signal from muscle electrodes. The amputee exerts muscle effort to counter the external forces. These devices are controlled in a natural way. The users use their own muscles to control them just as they would with their own arm [20].

In early 1970s the attention was turned to replace the human operator with a computer and soon important problems emerged. The operator is expected to see, hear, use his hands, understand, etc. The sensing and decision making capability that the humans use, had to be described in an efficient way to the computer.

Efforts to improve the performance of the robots in precise manipulations included both redesigning the arm and the robot control methods.Due to the imperfect model of the arm kinematics, deflection, forces, and inertia, the robot arm has a low positioning accuracy compared with many assembly tasks. Many methods have been proposed to improve the accuracy of the robot including new efficient computation schemes of the dynamics and gravity loading. Asada [21] developed the direct-drive arms in which each joint is connected to a rare-earth DC torque motor. The direct drive solution eliminates the back-lash generated by the power transmission, and a

better positioning can be obtained.

Even if a high arm positioning accuracy is obtained, it is still difficult to match it with the accuracy required by the high-tolerance mechanical assembly tasks. The solution for such problems includes both force and position feedback to modify the trajectories and to comply with the task constraints. D.E.Whitney [15,17], R.Paul [22], J.K.Salisbury [23], J.J.Graig, T.Mason [24] and many other investigators have conducted research on force control techniques. They have come up with a broad variety of different strategies. The rest of this section focuses on three different strategies which represent a milestone in this field.

Explicit feedback is looking for a linear relation between the end-effector force-torque and the end-effector position and orientation.

Hybrid control is the control of the position along specified degrees of freedom and the force-torque along the remaining degrees of freedom.

Robotic wrist which consists of mounting a wrist with active or passive compliance at the end of the manipulator.

3.4.1 Explicit Feedback

The explicit feedback method is based on the idea of generalized stiffness. Stiffness is defined [23] as the rate at which forces and torques on the

hand increase as it is deflected from a nominal position:

$$F = K\delta x \tag{3.5}$$

Where δx is the displacement, F is a vector of forces and torques, K is the stiffness matrix. Consider the case of inserting a peg in a hole. The position and forces are defined in a frame attached to the end of the peg. This frame is to follow a straight trajectory down the center of the hole and complying with forces along and torques around the x and y-axis. The K matrix is then [25]:

$$K = \begin{pmatrix} K_{soft} & 0 & 0 & 0 & 0 & 0 \\ 0 & K_{soft} & 0 & 0 & 0 & 0 \\ 0 & 0 & K_{hard} & 0 & 0 & 0 \\ 0 & 0 & 0 & K_{soft} & 0 & 0 \\ 0 & 0 & 0 & 0 & K_{soft} & 0 \\ 0 & 0 & 0 & 0 & 0 & K_{hard} \end{pmatrix},$$

The displacements along the axes with stiffness K_{hard} are position-controlled; the displacement along axes with stiffness K_{soft} are force-controlled. The choice of these numbers depends on the size of acceptable forces and position errors. This example is the most straightforward illustration of an explicit feedback control. The following is the analysis and result of the work done by Salisbury [23].

From equation 3.2 and the relation between the Cartesian displacement and the joint angle displacement ($\delta x = J\delta\theta$) we have:

$$F = KJ\delta\theta \tag{3.6}$$

Where J is the Jacobian matrix, $\delta\theta$ is an angular displacement.
The joint torque, T is needed to apply a Cartesian force; F, at the hand is defined by:

$$T = J^T F \tag{3.7}$$

Combining equations 3.3 and 3.4 we get the expression for joint torques to make the hand behave as a six-dimensional spring in the Cartesian space:

$$T = J^T K J \delta\theta \tag{3.8}$$

The term $J^T K J$ is called the joint stiffness matrix and denoted by K_θ. It should be noted that for equation 3.8 the position and force reference frame is fixed in the effector, and the stiffness matrix is diagonal. Since the Jacobian in equation 3.8 can be computed for any point, the force and position frame can be fixed at an arbitrary position and orientation relative to the hand. K_θ is a non-diagonal matrix, this implies that a position error in one joint has an effect on the commanded torque in all the other joints. Equation 3.8 is an expression of the joint torques that make the manipulator end-effector behave as a six degrees of freedom Cartesian spring.

The controller suggested by Salisbury to achieve the above result is:

$$T^c = J^T K J \delta\theta + J^T F_B \tag{3.9}$$

Where F_B is a bias forces.
A wrist force sensor is used to measure F_s which is used to determine the torque, (T_s), on the individual joints. The torque error, $(\delta T = T_c - T_s)$, is used to correct the applied motor torques to maintain the desired contact

force at the hand. Equation 3.10 is the torque applied to the ith joint [15].

$$T_i = T_{c,i} + G_i \delta T_i + K_{v,i} C_{II,i} \delta\dot{\theta}_i + V_{o,i} sgn(\dot{\theta}_i + C_{I,i}) \qquad (3.10)$$

Where:

$T_{c,i}$ = command torque, ith joint.
δT_i = torque error, ith joint.
$\dot{\theta}_i$ = velocity, ith joint.
$\delta\dot{\theta}_i$ = velocity error, ith joint.
G_i = torque compensation function, ith joint.
$K_{v,i}$ = velocity damping term, ith joint.
$C_{II,i}$ = instantaneous inertia, ith joint.
$C_{I,i}$ = gravity loading, ith joint.
$V_{o,i}$ = friction torque, ith joint.

Fig. 3.9 shows the block diagram of the control system described above.

3.4.2 Hybrid Control

This control method takes its origin in Mason's work [26]. Mason shows that a manipulator is partially constrained due to contact with one or more surfaces. Every manipulator task can be subdivided into subtasks

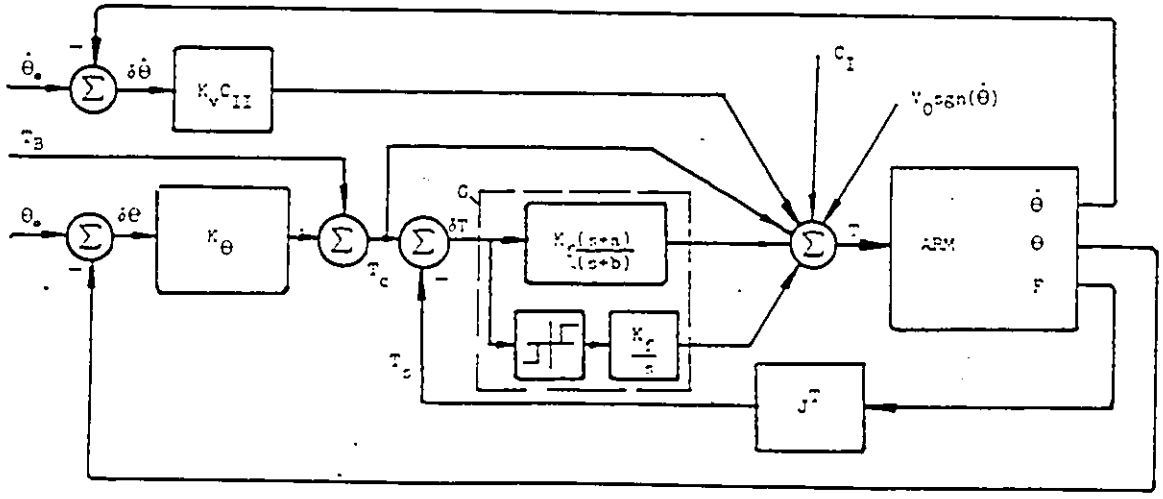


Figure 3.9: Stiffness control [23].

defined by a particular contact situation. A set of constraints called the *natural constraints* is associated to each subtask. The natural constraint can be of position type or force type. For example, if the manipulator is moving in free space the natural constraints are all force constraints. Since the manipulator has nothing to push against, all the forces are zero. If the end-effector is glued to something, the natural constraints are all position constraints. Then each task can be described in a frame called *constraint frame* {C} [22].

Using the same criteria an additional constraint called *artificial constraints* can be added to specify desired motion or force patterns. The artificial constraint is defined each time a user specifies a desired motion or

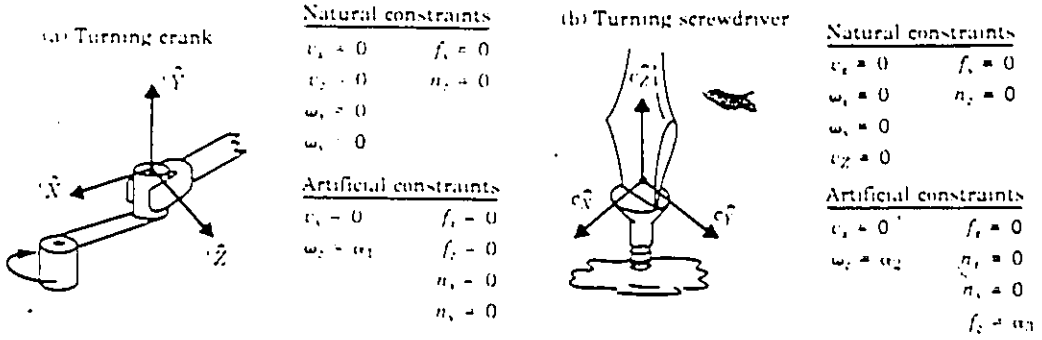


Figure 3.10: Natural and artificial constraints [26].

force. Fig 3.10 shows the natural and artificial constraints for two tasks.

The **hybrid position/force control** scheme is the case where degrees of freedom of the system are position controlled and others are force controlled. According to Graig [27] the hybrid controller must solve three problems:

1. Position control of a manipulator along direction in which a natural force constraint exists.

2. Force control of a manipulator along direction in which a natural position constraint exists.

3. A scheme to implement the arbitrary mixing of these modes along orthogonal degrees of freedom of an arbitrary frame, {C}.

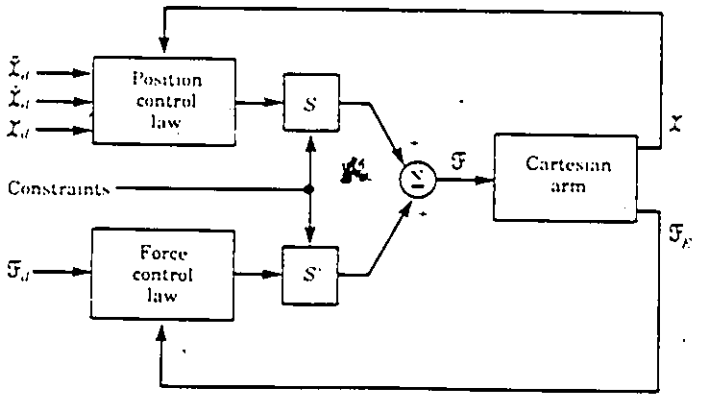


Figure 3.11: Generalized hybrid controller [27].

Figure 3.11 shows a generalized hybrid controller for a Cartesian arm. S and S' are diagonal matrices with zeros and ones. They control whichever mode (position or force) is to be used to control each joint. If a one is present in S, a zero is in S', and it is a position control mode. If a zero is present in S, a one is present in S', and it is a force control mode.

Raibert and Graig [28] described a hybrid controller by :

$$T_i = \sum_{j=1}^{N} \{\Gamma_{ij}(S_j \Delta F_j) + \psi_{ij}((1 - S_j)\delta X_i)\} \qquad (3.11)$$

where:

T_i = torque applied by ith actuator.

δF_j = force error in jth degree of freedom of (C).

δX_j = position error in jth degree of freedom of (C).

F_{ij} and ψ_{ij} = force and position compensation function, respec-

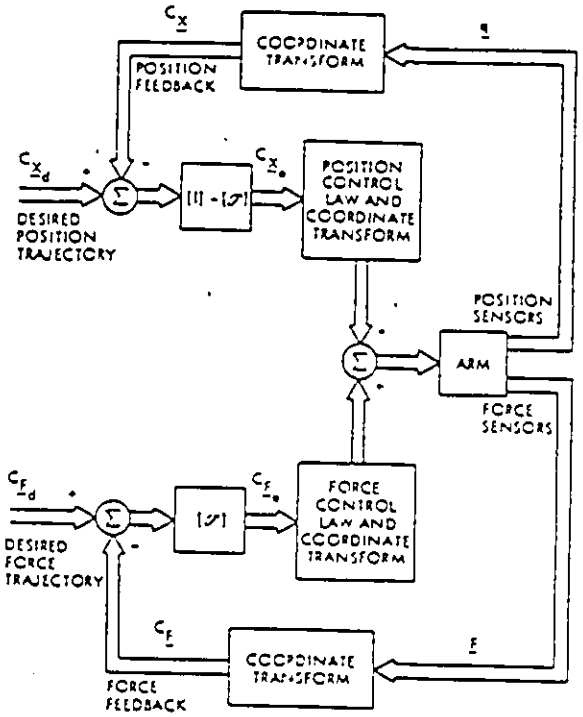


Figure 3.12: Conceptual organization of hybrid controller [28].

tively, for the jth input and ith output.

S_j = component of compliance selection vector.

The compliance selection vector, S, is N-tuple that work as S matrix described above. Figure 3.12 shows the structure of the hybrid controller.

3.4.3 Robotics Wrist

The techniques described above allow the robot to assemble parts. They work best when used with small and accurate manipulators. When a worker is assembling parts he uses the fingers and wrist for fine manipulation and leaves the arm and shoulder for the gross motions. To have such partitioning

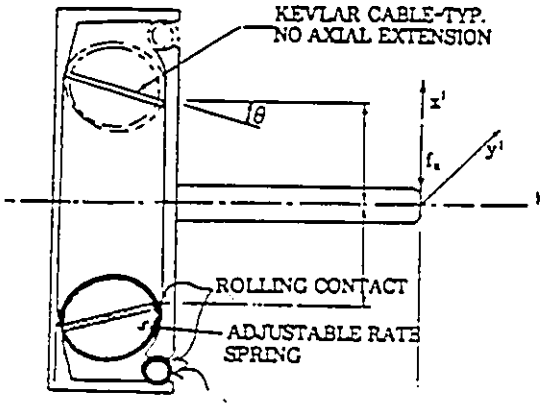


Figure 3.13: Side view of the compliant wrist [29].

of the task many researchers have suggested wrist devices which are often sandwiched between the gripper and the end of the arm. These devices can be found with active or passive compliance. The RCC wrist is a passive wrist used in assembly tasks. The RCC is extended for use in fine motion tasks other than assembly. The extended RCC is the instrumented RCC [30]. Deflection within the wrist are measured and used to modify the robot's path.

A robotic wrist based on the instrumented RCC wrist has been constructed [29,31]. This instrumented wrist can be automatically controlled for tasks with parts of different lengths and weights. Reinforced elastomeric pressurized spheres are used in the compliant platform to give the wrist a tunable stiffness. Figure 3.13 shows the instrumented, adjustable wrist. The base of the wrist is fixed to the end of a manipulator and the upper plate is fixed to the gripper.

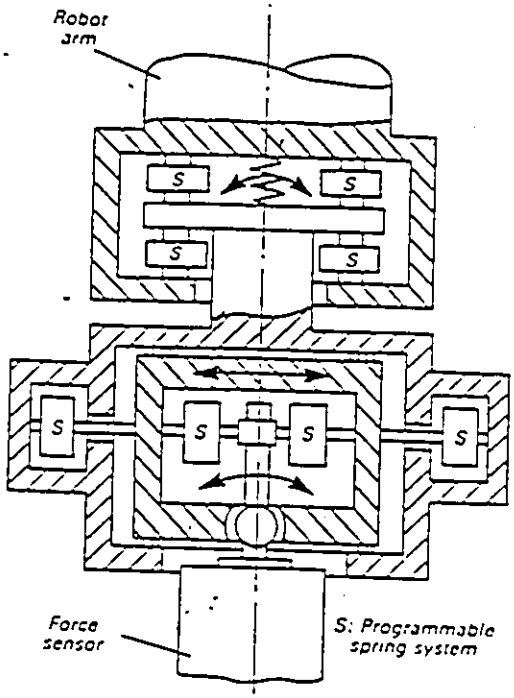


Figure 3.14: Programmable force controlled wrist [32].

Figure 3.14 shows a 5 degrees of freedom wrist. This wrist is microprocessor controlled, it has an adjustable stiffness. It is driven by a force-controlled soft servo system.

Chapter 4

Position Control

4.1 Introduction

In this chapter, both forward and inverse kinematic solutions of the robot arm are developed. The theory behind this solution as well as the notations used are briefly explained.

Unlike many manipulators,the solution of this problem is not part of the control system of the SIR-1 arm.This robot system comes with a primitive control method. The positioning of the end-effector is obtained through a set of move commands to each of the five motors. These commands are specifying the number of steps each motor has to perform. In this way, only an approximate information can be obtained on the position

or orientation in space of the end effector. In case of the "teach pendant" programming mode, the position or orientation may be evaluated visually by the robot's programmer. Under these circumstances programming the SIR-1 robot for complex moves and precise positioning, is a difficult if not impossible task.

Our work requires the ability to program the robot position in 3 D, Cartesian frame. Consequently kinematic transforms have to be developed.

4.2 Frames

In this section the working environment of the robot is described in terms of homogeneous coordinate transforms. A set of frames is first assigned to the workspace of the robot and then the different relations between the frames are obtained. The working area of the robot can be described by six frames as shown in Figure 4.1.

Base frame {B} is a nonmoving frame . In section 4.1.4, this frame is referred to as frame0.

Wrist frame {W} or the hand frame. It is attached to the last link of the robot and is a nonmoving frame.

Peg frame {P} generally refered to as the tool frame. It is attached to the end of the peg or tool that the robot is grasping.

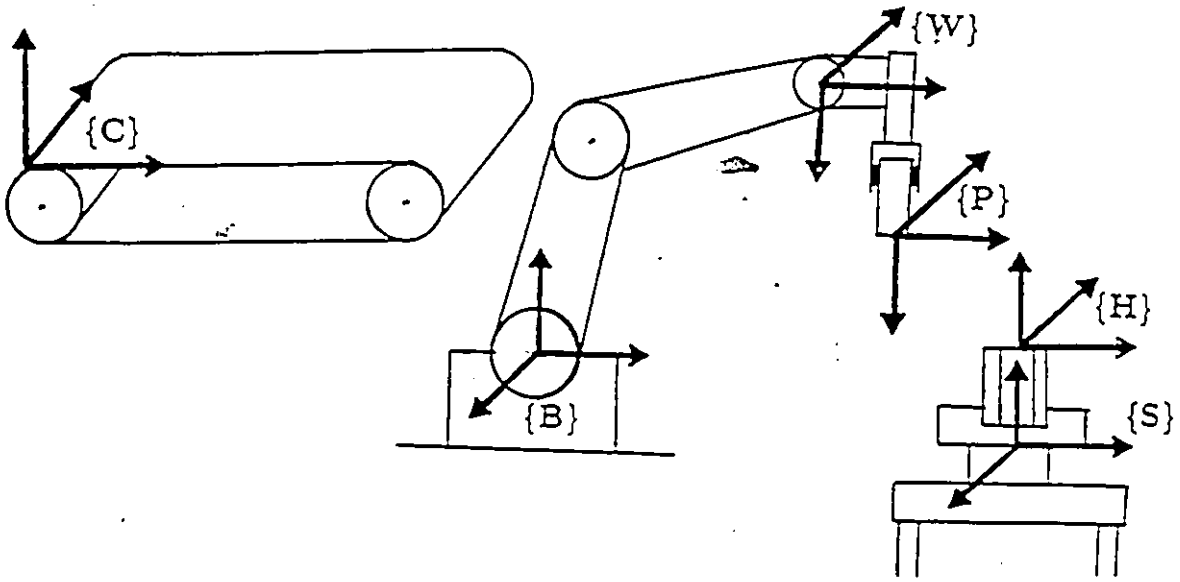


Figure 4.1: The robot workspace

Sensor frame {S} generally called the station frame. It is attached to the center of the sensor. The station frame is sometimes considered as the universe frame and all the robot moves are made relative to it. This frame is a nonmoving frame.

Hole frame {H} is also called the goal or target frame. It describes the location to which the robot is to move the peg.

Conveyor frame {C} can be considered as a station frame. It is attached to a nonmoving point of the conveyor. The object to be grasped is defined in this frame.

These frames are related by the following transforms:

T_P^W describes the frame at the peg {P} relative to the frame at the wrist {W}.

T_W^B describes the frame at the wrist {W} relative to the base frame {B}.

T_H^S describes the frame at the hole {H} relative to the sensor frame {S}.

T_S^B describes the frame at the sensor {S} relative to the base frame {B}.

T_P^H describes the frame at the peg {P} relative to the base hole {H}.

T_C^B describes the frame at the conveyor {C} relative to the base frame {B}.

From above we can write

$$T_P^B = T_W^B T_P^W \tag{4.1}$$

$$T_P^B = T_S^B T_H^S T_P^H \tag{4.2}$$

we can equate (4.1) and (4.2) to obtain

$$T_W^B T_P^W = T_S^B T_H^S T_P^H \tag{4.3}$$

we solve for T_W^B

$$T_W^B = T_S^B T_H^S T_P^H T_P^{W-1} \tag{4.4}$$

The goal is to make the peg frame coincide with the hole frame which means, in terms of transformation $T_P^H = I$ where I is the identity matrix. Equation (4.4) becomes

$$T_W^B = T_S^B T_H^S T_P^{W-1} \tag{4.5}$$

T_S^B, T_H^S and T_P^W is a constant matrices. In section 4.1.6 the inverse kinematics solution of the arm will be derived. Then given the above

matrix the arm joint angles that make frame {P} coincide with frame {H} can be found.

Once frame {P} and frame {H} have the same position and orientation (the peg is on the top of the hole) the control is switched to force control to resume the insertion operation.

4.3 Notation

Any robot arm can be described kinematically by four parameters for each link. These four parameters come in pairs: the link parameters which describe the link; The joint parameters which describe the link's connection to a neighboring link. The method used to define the different quantities follows the convention called **Denavit-Hartenberg notation** D-H. The four quantities are (figure 4.2) [33].

Link length , denoted by a_i. It is a distance from the intersection of the Z_{i-1} axis with the X_i axis to the origin of the ith frame along the X_i axis.

Link twist , denoted by α_i. It is an angle measured from the Z_{i-1} axis to the Z_i axis about the X_{i-1} in the right hand sense convention.

Link offset , denoted by d_i. It is the distance from the origin of the (i-1)th coordinate frame to the intersection of the Z_{i-1} axis with the X_i axis along the Z_{i-1} axis.

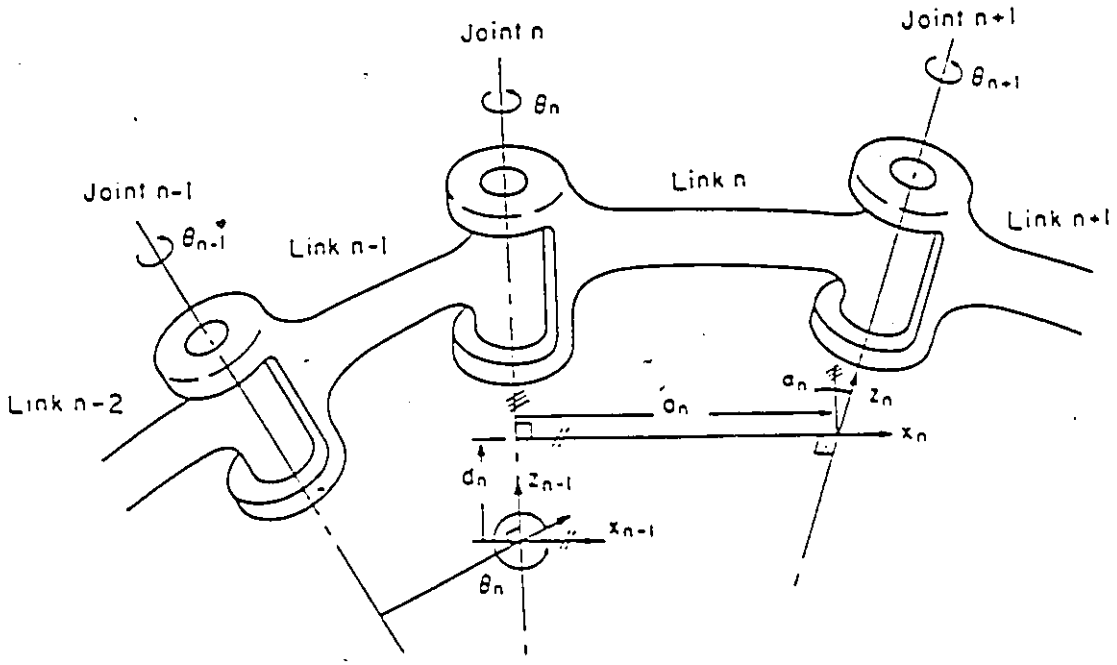


Figure 4.2: Link parameters and coordinate system [33].

Joint angle , denoted by θ_i. It is the angle from the X_{i-1} axis to the X_i axis about X_i.

For a rotary joint, the parameters are: d_i, a_i, and α_i. these parameters are fixed by mechanical design, while the joint angle θ_i changes when $link_i$ moves with respect to $link_{i-1}$. For a prismatic joint, the variable is d_i and the rest of the parameters are constants. Three rules have to be followed in assigning coordinate frames to the joints [34]. The joint coordinate systems were chosen in such a way that for a joint i:

- the Z_{i-1} axes lies along the axis of motion of the ith joint.
- the X_i axis is normal to the Z_{i-1} axis, pointing away from it.
- the Y_i axis complete the righthand coordinate system.

4.4 Link Transformation

The transformation that defines frame i relative to frame i-1 is a 4x4 homogeneous transformation matrix known as the D-H transformation matrix for adjacent coordinate frame:

$$A_i^{i-1} = Rot(Z_{i-1}, \theta_i)Tran(Z_{i-1}, d_i)Tran(X_{i-1}, a_i)Rot(X_i, \alpha_i) \quad (4.6)$$

$$A_i^{i-1} = \begin{pmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & aC\theta_i \\ S\theta_i & C\theta_i C\alpha_i & -C\theta_i S\alpha_i & aS\theta_i \\ 0 & S\alpha_{i-1} & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Where:

$$C\theta_i = \cos\theta_i$$

$$S\theta_i = \sin\theta_i$$

4.5 Kinematics of the SIR-1 Robot

The SIR-1 robot has five degrees of freedom. Figure 4.3 shows the assigning of link frames and table 4.1 shows the link parameters. In the case of our SIR-1 robot we have the following coordinate parameters:

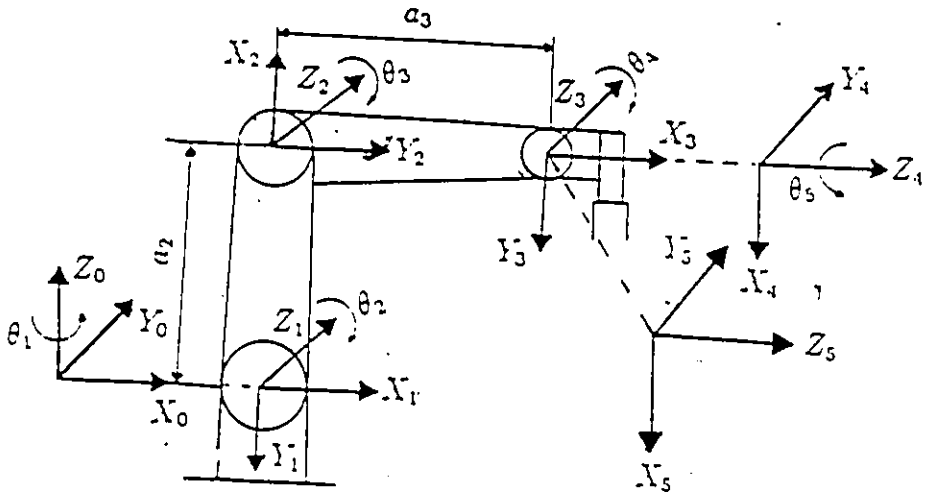


Figure 4.3: Link coordinate system for the SIR-1 robot

Table 4.1: SIR-1 arm link coordinate parameters.

Joint	θ_i	α_i	a_i	d_i	Range
1	θ_1	−90°	0	0	−150° to 150°
2	θ_2	0	250.0 mm	0	−110° to 20°
3	θ_3	0	204.0 mm	0	−55° to 5°
4	θ_4	90°	0	0	−90° to 90°
5	θ_5	0	0	0	350°

The five link transformation matrices for the SIR-1 robot are:

$$A_1 = \begin{pmatrix} C\theta_1 & 0 & -S\theta_1 & 0 \\ S\theta_1 & 0 & C\theta_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} C\theta_2 & -S\theta_2 & 0 & a_2C\theta_2 \\ S\theta_2 & C\theta_2 & 0 & a_2S\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} C\theta_3 & -S\theta_3 & 0 & a_3C\theta_3 \\ S\theta_3 & C\theta_3 & 0 & a_3S\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_4 = \begin{pmatrix} C\theta_4 & 0 & S\theta_4 & 0 \\ S\theta_4 & 0 & -C\theta_4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_5 = \begin{pmatrix} C\theta_5 & -S\theta_5 & 0 & 0 \\ S\theta_5 & C\theta_5 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4.5.1 Forward Kinematic Transform

By multiplying the A matrices together, we obtain the transform matrix that relates the position and orientation of the hand to the robot base frame (X_0, Y_0, Z_0). This transformation is a function of the joint angles.

The following are the intermediate results.

$$A_5^3 = A_4 A_5 = \begin{pmatrix} C\theta_4 C\theta_5 & -S\theta_5 C\theta_5 & S\theta_4 & 0 \\ S\theta_4 C\theta_5 & -S\theta_4 S\theta_5 & -C\theta_4 & 0 \\ S\theta_5 & C\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_5^2 = A_3 A_5^3 = \begin{pmatrix} C\theta_{34} C\theta_5 & -S\theta_5 C\theta_{34} & S\theta_{34} & a_3 C\theta_3 \\ S\theta_{34} C\theta_5 & -S\theta_{34} S\theta_5 & -C\theta_{34} & a_3 S\theta_3 \\ S\theta_5 & C\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_5^1 = A_2 A_5^2 = \begin{pmatrix} C\theta_{234} C\theta_5 & -S\theta_5 C\theta_{234} & S\theta_{234} & a_3 C\theta_{23} + a_2 C\theta_2 \\ S\theta_{234} C\theta_5 & -S\theta_{234} S\theta_5 & -C\theta_{234} & a_3 S\theta_{23} + a_2 S\theta_2 \\ S\theta_5 & C\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The final result is $A_5^0 = A_1 A_5^1$

$$A_5^0 = \begin{pmatrix} n_x & o_x & a_x & P_x \\ n_y & o_y & a_y & P_y \\ n_z & o_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Where the hand's parameters are:

$$n_x = C\theta_1 C\theta_{234} C\theta_5 - S\theta_1 S\theta_5 \quad (4.7)$$

$$n_y = S\theta_1 C\theta_{234} C\theta_5 + S\theta_1 S\theta_5 \quad (4.8)$$

$$n_z = -S\theta_{234} C\theta_5 \quad (4.9)$$

$$o_x = -C\theta_1 C\theta_{234} S\theta_5 - S\theta_1 C\theta_5 \quad (4.10)$$

$$o_y = -S\theta_1 C\theta_{234} + S\theta_5 C\theta_1 C\theta_5 \quad (4.11)$$

$$o_z = S\theta_{234} S\theta_5 \quad (4.12)$$

$$a_x = C\theta_1 S\theta_{234} \quad (4.13)$$

$$a_y = S\theta_1 S\theta_{234} \quad (4.14)$$

$$a_z = C\theta_{234} \quad (4.15)$$

$$P_x = a_3 C\theta_1 C\theta_{23} + a_2 C\theta_1 C\theta_2 \quad (4.16)$$

$$P_y = a_3 S\theta_1 C\theta_{23} + a_2 S\theta_1 C\theta_2 \quad (4.17)$$

$$P_z = -a_3 S\theta_{23} - a_2 S\theta_2 \quad (4.18)$$

Matrix A_5^0 defines the position and the orientation of the hand with respect to the base frame.

The first three columns define three orthogonal vectors which specify the hand orientation. The third column is the approach vector which

points in the direction of the wrist. The second column is the orientation vector, which, with the approach vector, define the orientation of the hand. The first column is the normal vector which is orthogonal to the other two. The fourth column is the position of the hand frame. Then, the position, in the cartesian frame, of the hand is :

$$\begin{aligned} X &= C\theta_1(a_3C\theta 23 + a_2C\theta_2) \\ Y &= S\theta_1(a_3C\theta_{23} + a_2C\theta_2) \\ Z &= -a_3S\theta_{23} - a_2S\theta_2 \end{aligned}$$

The orientation of the hand will be discussed later.

By having X,Y,Z a function of θ_1, θ_2 and θ_3 the angles of the base, the shoulder and the elbow, the forward kinematics problem is solved. Given the amount of deviation of the base,the shoulder, and the elbow, it is possible to evaluate the position of the hand in the Cartesian.

4.5.2 Inverse Kinematic Transform

The forward kinematic transform is a transform from the joint space to the Cartesian space. The inverse kinematic transform is a transform from the Cartesian space to the joint space. To solve the inverse kinematic problem we assume that a position in space is known and we find the joint angles corresponding to that position.

Now, given the numerical value of A_H^B, a transform relating the hand frame to the base frame, one can finds the values of $\theta_1, \theta_2 \ldots \theta_5$. In

this case there are twelve equation and five unknowns. However, only two out of nine equations of the A_5^0 orientation parts are independent. These two added to the three position equations give six equations with five unknowns. These equations are nonlinear. There is no general algorithm which can solve a nonlinear equation. The mechanical design of the manipulator is such that the α_i are 0 or $\pm$90degrees and many link offsets are 0. This fact guarantees the existence of the solution and makes it easier than it appears.

The hand configuration is described by

$$A_H^B = A_1 A_2 A_3 A_4 A_5$$

where A_H^B is assumed to be known.

We premultiply by A_1^{-1}

$$A_1^{-1} A_H^B = A_2 A_3 A_4 A_5$$

The individual elements of the matrix are equated then inspected for a solution for one or more angles using the ANTAN2 function. For the rest of the angles we can obtain another set of equations by premultiplying by A_2^{-1}. Then, we keep premultiplying by the inverse of the A matrices until all the solutions are identified.

For the SIR-1, the inverse kinematic problem cannot be solved totally by the method described above. This happens because the SIR-1 has four parallel joints, and premultiplying by A_2^{-1}, A_3^{-1} and A_4^{-1} provides no useful information.

The solution developed further is a combination of an algebraic and geometric solution.

Assume

$$A_H^B = \begin{pmatrix} n_x & o_x & a_x & P_x \\ n_y & o_y & a_y & P_y \\ n_z & o_z & a_z & P_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_1^{-1}A_H^B = \begin{pmatrix} C\theta_1 n_x + S\theta_1 n_y & C\theta_1 o_x + S\theta_1 o_y & C\theta_1 a_x + S\theta_1 a_y & C\theta_1 P_x + S\theta_1 P_y \\ -n_z & -o_z & -a_z & -P_z \\ C\theta_1 n_y - S\theta_1 n_x & C\theta_1 o_y - S\theta_1 o_x & C\theta_1 a_y - S\theta_1 a_x & C\theta_1 P_y - S\theta_1 P_x \\ 0 & 0 & 0 & 1 \end{pmatrix} = A_5^1$$

From above we can identify :

$$C\theta_1 P_y - S\theta_1 P_x = 0 \Rightarrow \theta_1 = Atan2(P_y, P_x) \qquad (4.19)$$

$$S\theta_5 = -S\theta_1 n_x + C\theta_1 n_y; C\theta_5 = -S\theta_1 o_x + C\theta_1 o_y$$

$$\Rightarrow \theta_5 = Atan2(-S\theta_1 n_x + C\theta_1 n_y, -S\theta_1 o_x + C\theta_1 o_y) \qquad (4.20)$$

$$C\theta_{234} = a_z; S\theta_{234} = C\theta_1 a_x + S\theta_1 a_y$$

$$\theta_{234} = Atan2(C\theta_1 a_x + S\theta_1 a_y, a_z) \qquad (4.21)$$

Premultiplying by A_2^{-1}, A_3^{-1} or A_4^{-1} won't yield any useful equations. To find θ_2, θ_3, θ_4 the following approach is used: From figure 4.4 we get:

$$W^2 = P_z^2 + P_x^2 + P_y^2 \qquad (4.22)$$

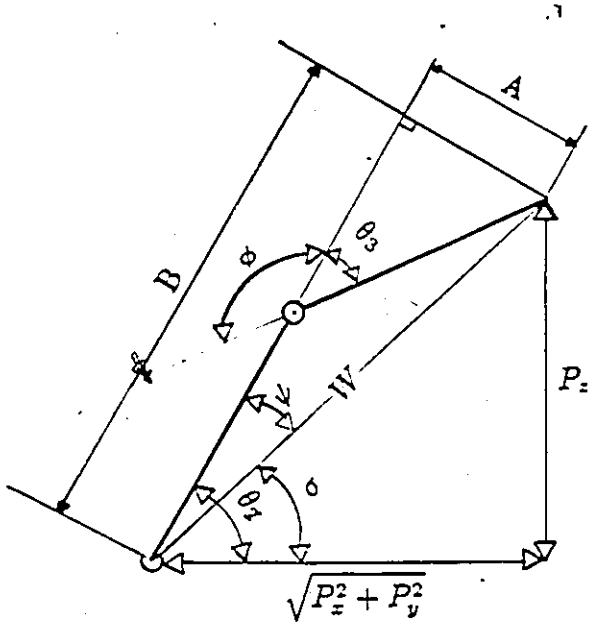


Figure 4.4: Links 2 and 3 of the arm

$$W^2 = a_2^2 + a_3^2 + 2a_2a_3\cos\phi \tag{4.23}$$

$$\theta_3 = \pi - \phi \Rightarrow \cos\theta_3 = \cos\phi \tag{4.24}$$

Combining the three equation 4.17, 4.18 and 4.19 we get

$$P_x^2 + P_x^2 + P_y^2 = a_2^2 + a_3^2 + 2a_2a_3\cos\theta_3 \tag{4.25}$$

which leads to

$$\cos\theta_3 = \frac{P_x^2 + P_y^2 + P_z^2 - a_2^2 - a_3^2}{2a_2a_3} \tag{4.26}$$

$$\sin\theta_3 = \pm\sqrt{1 - \cos^2\theta_3}$$

in the case of SIR-1 the elbow is always up which leads to : $\sin\theta_3 = \sqrt{1 - \cos^2\theta_3}$ Then we have :

$$\theta_3 = Atan2(\sqrt{1 - \cos^2\theta_3}, \cos\theta_3) \tag{4.27}$$

Also, from figure 4.3 we have $-\theta_2 = \psi + \sigma$

$$\sigma = Atan2(P_z, \sqrt{P_x^2 + P_y^2}) \qquad and \qquad \psi = Atan2(A, B)$$

with $A = a_3 \sin\theta_3$ and $B = a_3 \cos\theta_3 + a_2$ then

$$\theta_2 = Atan2(P_z, \sqrt{P_x^2 + P_y^2}) - Atan2(a_3 \sin\theta_3, a_2 + a_3 \cos\theta_3) \quad (4.28)$$

θ_4 is obtained by :

$$\theta_4 = \theta_{234} - \theta_2 - \theta_3 \quad (4.29)$$

The complete inverse kinematic solution for the SIR-1 arm is obtained. This solution assumes the following quantities to be numerically known:P_x, P_z, n_x, o_x, o_y, a_x, a_y, a_z. The first three values define the hand frame position with respect to the base frame, their numerical values are given by the user. The rest of the quantities define the orientation of the hand frame with respect to the base frame. Their numerical values are obtained indirectly through the orientation angles given by the user. The next section shows how these quantities are obtained.

4.5.3 Hand Orientation

The orientation of the hand is given by the rotation part of A_H which is :

$$R_H = \begin{pmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{pmatrix}$$

One usual way of describing the orientation of the hand frame with respect to the base frame is represented by the *Roll, Pitch, Yaw* (RPY) angles [27]. We start with frame H coincident with frame B. First rotate H about X_H by an angle γ, then rotate about Y_H by an angle β and then rotate about Z_B by an angle α.

The final expression of the rotation is :

$$R_{RPY} = Rot(Z_H, \alpha)Rot(Y_H, \beta)Rot(X_H, \gamma) \tag{4.30}$$

$$R_{RPY} = \begin{pmatrix} C\alpha & -S\alpha & 0 \\ S\alpha & C\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} C\beta & 0 & S\beta \\ 0 & 1 & 0 \\ -S\beta & 0 & C\beta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & C\gamma & -S\gamma \\ 0 & S\gamma & C\gamma \end{pmatrix}$$

$$R_{RPY} = \begin{pmatrix} C\beta C\alpha & C\alpha S\beta S\gamma - S\alpha C\gamma & C\alpha S\beta C\gamma + S\gamma S\alpha \\ C\beta S\alpha & S\gamma S\beta S\alpha + C\gamma C\alpha & S\beta C\gamma S\alpha - S\gamma C\alpha \\ -S\beta & C\beta S\gamma & C\beta C\gamma \end{pmatrix}$$

For the case of SIR-1 , with 5 degrees of freedom, $\gamma = 0$:

$$R_{RPY} = \begin{pmatrix} C\beta C\alpha & -S\alpha & S\beta C\alpha \\ C\beta S\alpha & C\alpha & S\beta S\alpha \\ -S\beta & 0 & C\beta \end{pmatrix}$$

Given α and β, the roll and pitch angles, The orientation of the hand can be calculated.

The inverse problem is straight forward. Assume the orientation of the hand is known by the rotation matrix R_H α and β can be identified

as :

$$\alpha = Atan2(-o_x, o_y) \tag{4.31}$$

$$\beta = Atan2(-n_z, a_z) \tag{4.32}$$

4.6 Implementation of the Kinematic Model

The time required to perform kinematic calculation can be important. When implementing the kinematic model of the SIR-1 robot arm, some techniques have been used to minimize the time. By factoring the equations, it is possible to reduce the number of multiplications and additions. By creating local variables it is possible to avoid calculating common terms over and over again. The calculation of the transcendental functions (i.e. sine and cosine) is very time consuming. Using an 80S7 math coprocessor reduces the time required to calculate a sine or cosine.

Figure 4.5 shows a block diagram of the kinematic model implementation of the SIR-1 robot arm. This implementation has 3 levels above the robot system hardware:

I/O - This level converts the data to a format that can be accepted by the robot controller and sends it to the controller. It also receives the data from the controller and converts it to data that can be manipulated by the rest of the program. The source code of the program can be found in Appendix E.

Robot utility - This level includes the implementation of the command set and the kinematic model of the SIR-1 robot arm. It also reports any errors occurred during the calculation or from the I/O port. The source code of this level can be found in Appendix C.

User - This level can be designed by any user wishing to use the SIR-1 robot. In Appendix D, we included two such programs: The first one takes as parameters the cartesian coordinates of a given point and moves the robot to that point. The second program controls the opening of the gripper. These programs can be conceived as separate modules and linked together when needed.

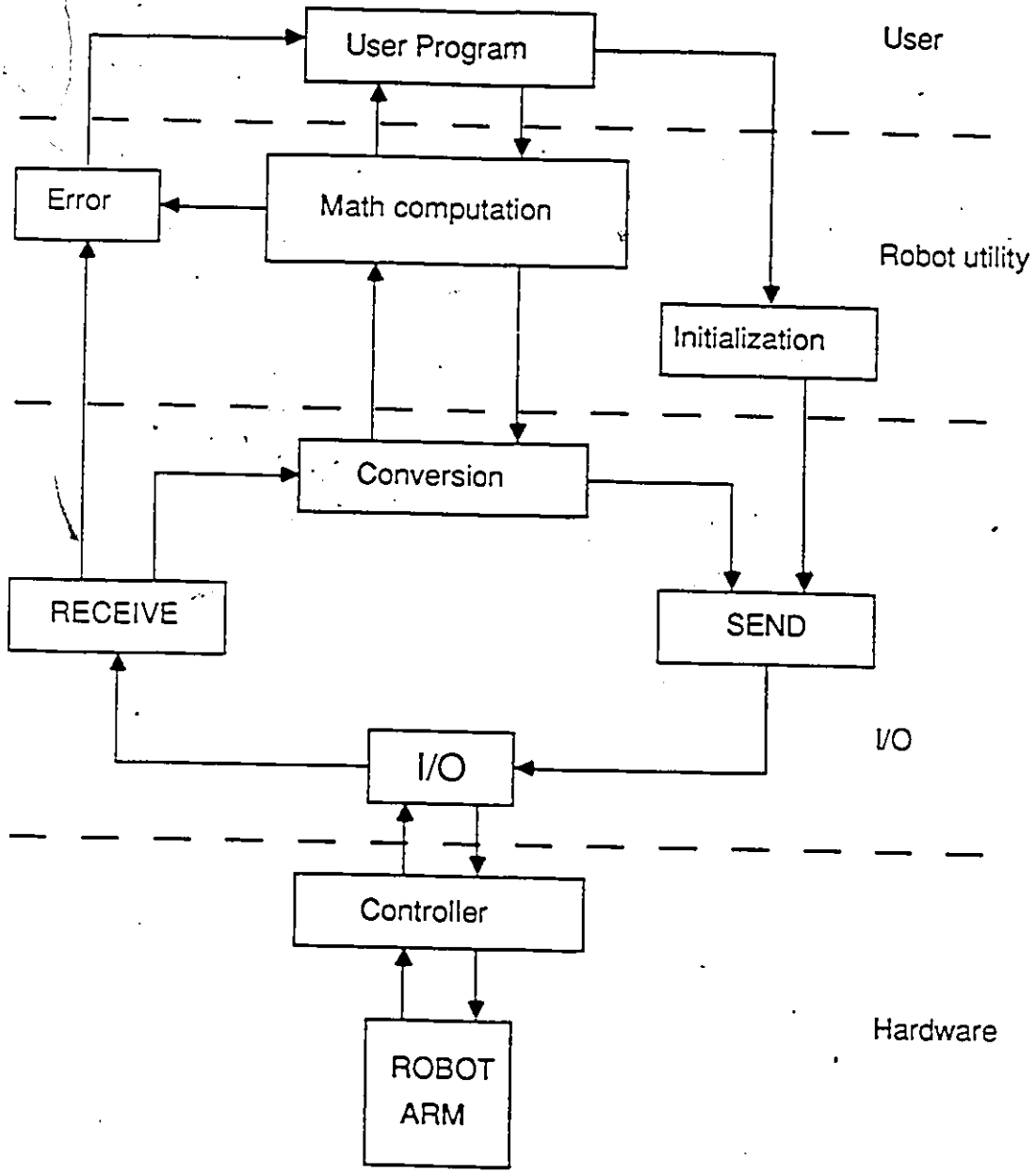


Figure 4.5: Block diagram of the implementation of the SIR-1 robot arm kinematic model

Chapter 5

Force Control: Analysis, Implementation and Performance

5.1 Introduction

In the previous chapter the position control of the robot system was discussed. For an ideal robot arm and an ideal control system, a position control would be enough for a robot arm to accomplish simple parts fitting tasks. However, because of many limiting conditions, it is extremely difficult to employ a position control robot in a delicate operation such as parts mating. Some of these limiting conditions are as follows:

- The working environment is highly unpredictable.
- The required high relative positional accuracy of the parts.
- The actual mechanical error in the robot.
- The robot control error (e.g. due to quantization, modeling, etc.)

Therefore an alternative control mechanism using a compliance device should be used to compensate for these positioning errors.

In this chapter an active force control algorithm is presented. This algorithm is conceived for the peg-in-hole operation because it is by far the most used type of operation in mechanical assembly. Even though only active force control is discussed, a passive compliance exists due to the imperfect mechanical construction of the robot arm. While this mechanical limitation is considered as a major disadvantage for the positioning control, it is an advantage when dealing with force control. The analysis of this built-in passive compliance is not within the scope of this work.

As mentioned in the previous chapter a force/torque sensor is located at the fixture. Force servoing is performed at the Cartesian level which eliminates the need for computing the Jacobian.

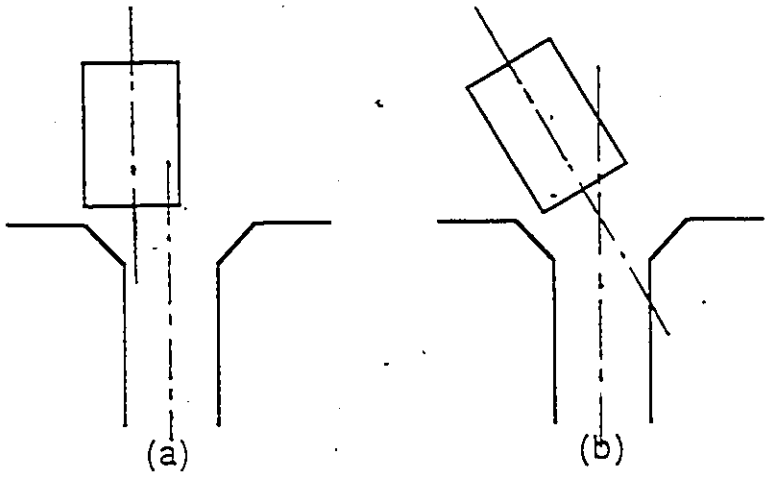


Figure 5.1: positional errors (a) lateral error, (b) angular error.

5.2 Types of errors and their consequences

Prior to insertion, two positional errors may occur, a lateral error and an angular error. A small lateral error will result in an angular error, see figure 5.1. A large lateral error will prevent the peg from entering the hole. The angular error is considered to be small so that the peg can slide in the hole during one point contact.

The critical phase, during the insertion motion, is the occurrence of two points contact. At this stage two unwanted phenomena may occur: either wedging or jamming (defined in section 3.3.1). Equation (3.3) and (3.4) can be used to prevent jamming. Wedging can be a severe problem that once it occurs, any attempt to proceed the insertion will cause damage to the parts or to the robot arm. Figure 5.2 shows the forces during wedging; the two contact forces f1 and f2 point directly toward each other

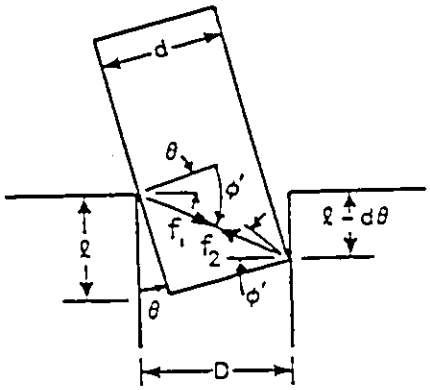


Figure 5.2: forces during wedging [17].

[17]. Wedging can then be detected when Fz becomes very large while the lateral forces are zero. The only remedy for wedging is to pull the peg out and to try again. Jamming is a consequence of the relation between the insertion forces and the frictional forces. It can be corrected by changing the direction of the applied forces.

5.3 Active Force Control

5.3.1 Force Reading

From the instruction set of the ASTEK FS-100-A (appendix B) we have developed and implemented a "C" program library having the two following functions:

i) interface between the sensor and the computer.

ii) interface between the computer and the programmer.

For real time consideration the interface between the sensor and the computer is implemented using the S6/SS assembler. The interface between the computer and the programmer is implemented in C language. The listing of the library programs can be found in appendix E & F. An A/D converter is used to read the analog output of the sensor. The three forces and three moments can be accessed from a C program by Fx, Fy, Fz, Mx, My and Mz. The forces are in [N] and the moments are in [N.M].

The forces and torques are measured at the sensor frame. Since the sensor is located at the fixture and away from the wrist a change of frame is needed. This frame change is needed to get the equivalent forces and torques at the wrist level. Let $(Fx_C, Fy_C, Fz_C, Mx_C, My_C, Mz_C)$ be the forces at the sensor frame and $(Fx_S, Fy_S, Fz_S, Mx_S, My_S, Mz_S)$ be the equivalent forces at the wrist along the sensor frame and not the base frame, see Figure 5.3. Let (X,Y,Z) be the distance between the origin of the sensor and the origin of wrist frame. We have then the matrix transform:

$$\begin{pmatrix} Fx_S \\ Fy_S \\ Fz_S \\ Mx_S \\ My_S \\ Mz_S \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -Z & Y & 1 & 0 & 0 \\ Z & 0 & X & 0 & 1 & 0 \\ -Y & X & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} Fx_C \\ Fy_C \\ Fz_C \\ Mx_C \\ My_C \\ Mz_C \end{pmatrix}$$

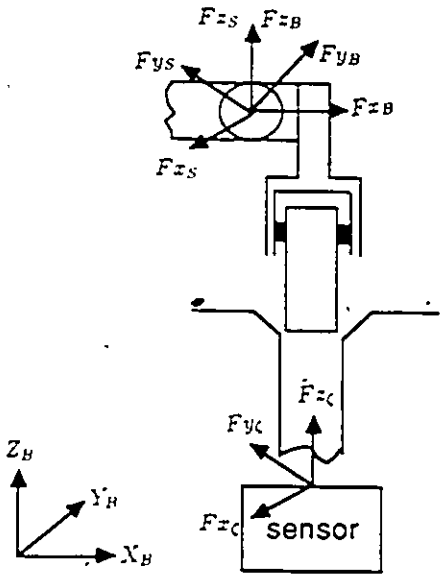


Figure 5.3: Force reading

This transformation, described above, is performed by the inboard microprocessor. This procedure involves the communication of X,Y and Z to the sensor. As the peg slides in the hole the value of Z has to be updated.

The next step is to have the forces with respect to the base frame. Let R_S^B be the rotation part of T_S^B, defined as a matrix:

$$R_S^B = \begin{pmatrix} r_{11} & r_{21} & r_{31} \\ r_{12} & r_{22} & r_{32} \\ r_{13} & r_{23} & r_{33} \end{pmatrix}$$

Let F_B be the force vector at the wrist along the base frame and F_S the force vector at the wrist along the sensor frame, see Figure 5.3. We have

then:

$$\begin{pmatrix} Fx_B \\ Fy_B \\ Fz_B \\ Mx_B \\ My_B \\ Mz_B \end{pmatrix} = \begin{pmatrix} r_{11} & r_{21} & r_{31} & 0 & 0 & 0 \\ r_{12} & r_{22} & r_{32} & 0 & 0 & 0 \\ r_{13} & r_{23} & r_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & r_{11} & r_{21} & r_{31} \\ 0 & 0 & 0 & r_{12} & r_{22} & r_{32} \\ 0 & 0 & 0 & r_{13} & r_{23} & r_{33} \end{pmatrix} \begin{pmatrix} Fx_S \\ Fy_S \\ Fz_S \\ Mx_S \\ My_S \\ Mz_S \end{pmatrix}$$

For practical consideration, the sensor and the part are located such that :

$$R_S^B = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The sensor is factory calibrated. A reset bias command (RB) is provided to zero the sensor reading. If the RB is given before the peg touches the hole the reading will be that of the reaction forces and not of the fixture loading.

5.3.2 Fine motion

Once the peg is in contact with the hole, even a small move in the wrong direction will result in large contact forces. It is essential to provide the robot arm with a fine motion control before attempting to implement any force control strategy. The idea behind the fine motion control, used in this work, can be defined by the following criteria:

Table 5.1: Look-up table for fine motion

direction \ Joint		1	2	3	4	5	Cartesian
X-axis	$-\Delta X$	0	-3	1	0	0	-0.3 mm
	ΔX	0	3	-1	0	0	0.3 mm
Y-axis	$-\Delta Y$	1	0	0	0	0	-1.0 mm
	ΔY	-1	0	0	0	0	1.0 mm
Z-axis	$-\Delta Z$	0	-1	-5	0	0	-0.5 mm
	ΔZ	0	1	5	0	0	0.5 mm
Pitch	$-\Delta\beta$	0	0	0	1	0	$-0.1S^{\circ}$
	$\Delta\beta$	0	0	0	-1	0	$0.1S^{\circ}$
Roll	$-\Delta\alpha$	0	0	0	0	1	$-0.1S^{\circ}$
	$\Delta\alpha$	0	0	0	0	-1	$0.1S^{\circ}$

- The motion of the end-effector is divided into three translations along the Cartesian axes and 2 rotations.

- The end-effector will perform only one translation or one rotation at a time and not a combination of the two.

- The size of the translation or the rotation is to be the minimum possible value (example 0.3 mm x-dir).

To minimize the computing time the different fine motions are translated from Cartesian to joints values. A look-up table is constructed by simulating the arm kinematics around the sensor position, see table 5.1. In

this way the inverse kinematics is not computed during the force control process. The joints values are obtained by moving in one direction by 10 mm for a translation and 10° for a rotation, while keeping the rest of the directions constants. The equivalent joints values are then divided by their greatest common divider.

5.3.3 Force Algorithm

Figure 5.4 shows a simplified flowchart of the force control algorithm.

During the peg-in-hole insertion the following threshold values are used by this algorithm Fzth1, Fzth2, Fxth, Fyth, Mxth, Myth.
Fzth1, Fxth, Fyth, Mxth, and Myth are the force values used to detect jamming or an excessive built up of forces.
Fzth2 is the force level below which after a lateral correction the insertion is restarted.

A first version of the insertion algorithm consists of the following steps:

1. Move the peg in Z+ direction until a threshold force (Fzth1) is obtained or the desired position is reached;
2. If desired position is reached then stop;
3. If Fz becomes bigger than Fzth1 then a two point contact has occurred,

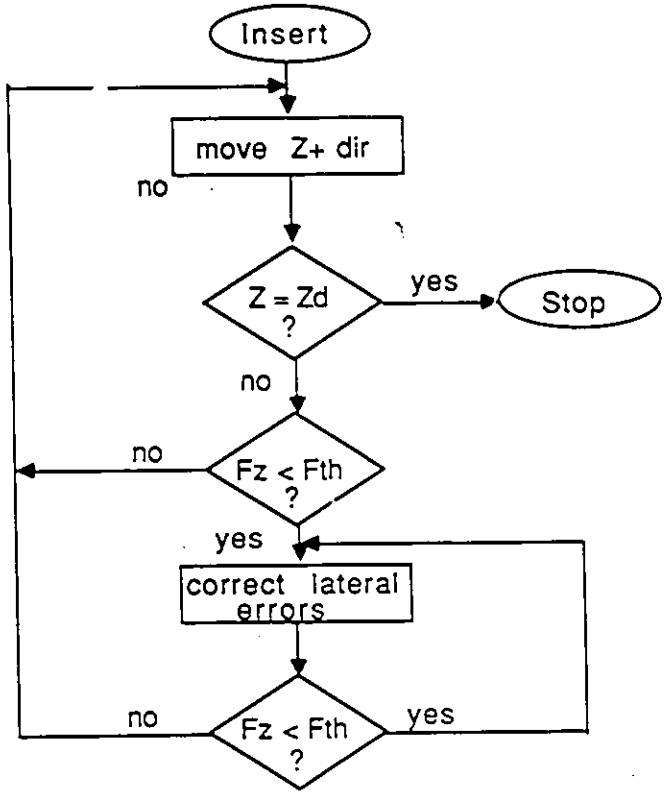


Figure 5.4: Force control flowchart

and the result is a jamming. Go to step 4;

4. The lateral forces and torques are read, and a move in the appropriate direction to minimize the lateral force is initiated;

5. Step 3 is repeated until Fz becomes less than Fzth2;

6. The process starts over again by going to step 1;

When implemented, this algorithm suffered from two major problems:

- Because the jamming situation is solved by trial-and-error this process

sometimes takes an enormous amount of time before it is resolved. This is mainly due to the relatively large moves of the end-effector.

- Even more frequently than jamming, The wedging may occur. This situation is characterized by a large value of Fz and small values of the lateral forces. In the wedging case the program is trapped in an infinite loop. This does not come as a surprise if we consider equation 3.2 ($\theta_2 < \frac{C}{\mu}$), θ_2 being the allowable angular error to avoid wedging. For C=0.001 and $\mu = 0.5$, θ_2 should be less than 0.22°. It is hard to meet this condition with the robot arm used in this work.

To correct these problems some changes have to be made. When detected, both problems are treated in the same way and they are refered to as wedging. There were no changes in the previous algorithm from step 1 to step 3; from step 4 on, the changes are as follows:

4. Step 3 is repeated until lateral forces become small or the maximum number of repetition is reached.
5. If Fz becomes less than Fzth2, go to step 1.
6. If lateral forces are small or maximum repetition reached and Fz is larger than Fzth1, then correct the wedging.
7. Go to step 1.

The detailed flowchart of the new algorithm described above is illustrated in figure 5.5. Figure 5.6 shows the corresponding pseudo code. The listing

of the computer program is given in appendix G. The force threshold values are computed using equation 3.3 and 3.4. The lateral correction includes 4 motions, 2 translations and 2 rotations. The first translation is along the x-axis and the second is along the y-axis. The first rotation is around the x-axis and the second is around the y-axis. Initially, these 4 corrections were done separately, which sometimes resulted in the following dilemma: a correction of the force along one axis will result in a reaction torque around the orthogonal axis. The correction of the torque will result in a reaction force along the orthogonal axis. To solve this problem the following strategies were successfully implemented:

- First, the reaction forces (Fx and Fy) are corrected, then the reaction torques (Mx and My). In this way, any torques resulting from the force correction will be taken into account.
- Due to the angular correction, a force tends to arise. This is overcome by the simultaneous torque and force correction.

Since a correction along (around) a given axis does not result in an error along (around) the orthogonal axis, a further shortcut can be obtained by correcting the two forces or two torques simultaneously.

The wedging situation is corrected in a natural way by imitating a human worker performing the same task. This solution is based on two steps:

1. The peg is moved along a direction opposite to the insertion direction.

2. The peg is rotated by a small angle back and forth to simulate a shaking movement.

These two steps are repeated until Fz becomes small. If, during this process, the peg leaves the hole, the insertion operation is aborted, and the assistance of the operator is needed. However, this situation is unlikely to occur.

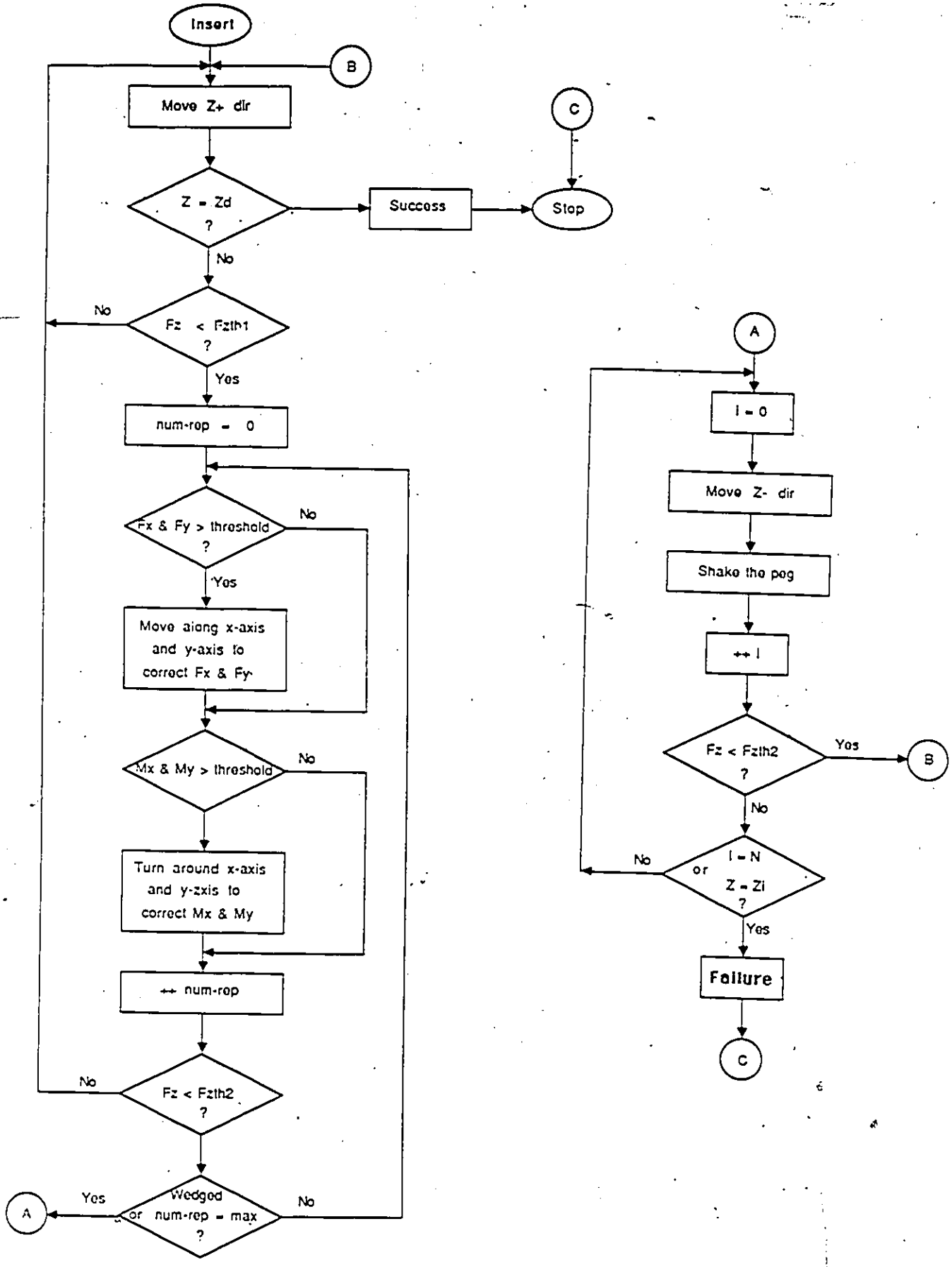


Figure 5.5: detailed force control flowchart

```
Insert
BEGIN
    Z, Zd :actual and desired position;
    num-rep :number of repetition;
    max :maximum number of repetition;
    WHILE    Z < Zd    DO
        WHILE    Fz < Fzth1 and Z < Zd    DO
            move (dir(+Z));
        IF    Z > Zd    THEN
            stop;
        num-rep=0;
        WHILE    Fz > Fzth2 and repeat < max    DO
            IF    Fx > Fxth or/and Fy > Fyth    THEN
                correct lateral error;
            IF    Mx > Mxth or/and My > Myth    THEN
                correct angular error;
            ++num-rep;
        IF    repeat = max or Fx=Fy=0 and Fz > Fzth2    THEN
            correct the jamming;
END
```

Figure 5.6: Insert operation

5.4 Results and Discussion

5.4.1 Set-up

To test the algorithm proposed in this chapter a simple set-up is used. The fixture is a rigid table on which the sensor is fixed. The part with the hole is fixed on top of the sensor. The peg and the hole are tightly fit (Fig. 5.7).

Two peg-in-hole operation having different clerences are performed. The dimensions corresponding to these operation are as follow:

operation 1:

D = 25.460 mm

d = 25.410 mm

l = 45 mm

C = 0.001

operation 2:

D = 25.460 mm

d = 25.427 mm

l = 45 mm

C = 0.001

D is the diameter of the hole, d is the diameter of the peg, l is the length of the peg and C is the clearance ratio. The two clearance ratios used through this experiments are comparable to those used in precise mechanical assembly. It has been reported that problems occurs only at the beginning of

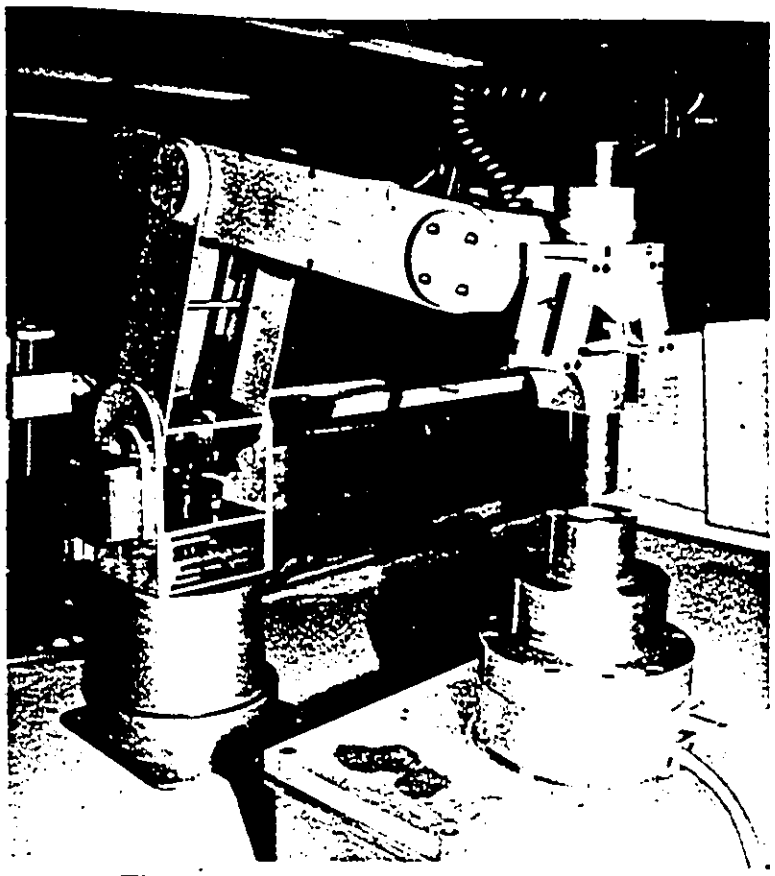


Figure 5.7: Photograph of the set-up

the insertion operation. This fact was also observed during this experiment. For this reason the insertion depth is limited to 20 mm. After 20 mm of insertion the operation will resume normally without any significant problems.

5.4.2 Performance Analysis

Two types of results are analyzed, the time history of the contact forces and the evolution of Fz with depth. It should be noted that two different experiments using the same parts will yield two different results. However, all results will have a common overall behavior. From this behavior one can recognize the different steps of the algorithm.

Shown in Figure 5.8 are the time histories of the measured contact forces during a typical insertion operation. These results correspond to a clearance ratio of 0.002.

Contact between the peg and the hole is made when Fz drops (at t < 0.25 sec). As the threshold force is not reached, the insertion operation proceeds until around t = 10 sec where there is a build up of force. After a lateral correction along the x-axis and an angular correction around the x-axis, the operation is resumed. For a tighter fit the build up of forces is more frequent which slows down the operation. This fact can be seen in Figure 5.9 which represents the history of the forces when the clearance ratio is 0.001.

No major changes in magnitude of Fy and Mx can be seen. This is due to the accidental passive compliance built in the robot arm. An inspection of the 5 joints of the arm reveals that joint 1 (base joint) and joint 5 (roll joint) have an important backlash, the rest of the joints are much stiffer. These two joints have an important effect on the arm move in the y-direction (joint 1) and rotation around the x-axis (joint 5). This fact proves again the advantage of having a passive compliance in assembly tasks.

All the insertion steps can be seen from the time history graphs. However the wedging situation is not clearly seen from these graphs, plotting Fz versus the depth gives useful information about the wedging situation. From Figure 5.10, at depth around 2.5, 7.8, 12.5, and 17.5 sec, the effect

of the wedging correction is shown. The behavior of the graph at these points is due to the fact that when the wedging occurs the peg is pulled in the Z+ direction until Fz becomes small enough to proceed with insertion. Figure 5.11 shows the case for an operation where no wedging has occurred.In Figure 5.12 the initial errors are large, as a result, a severe wedging has occurred. This wedging has occurred around a depth of 2.5 mm. After a few trial the wedging situation was solved and the insertion was resumed with no further wedging. Due to the poor stifness of the robot arm a relativelly large error occurs prior to the mating phase. This resulted in an aproximate figure of 15% of failure rate. Once the peg is correctly positioned the insertion phase is sucessfully carried out.

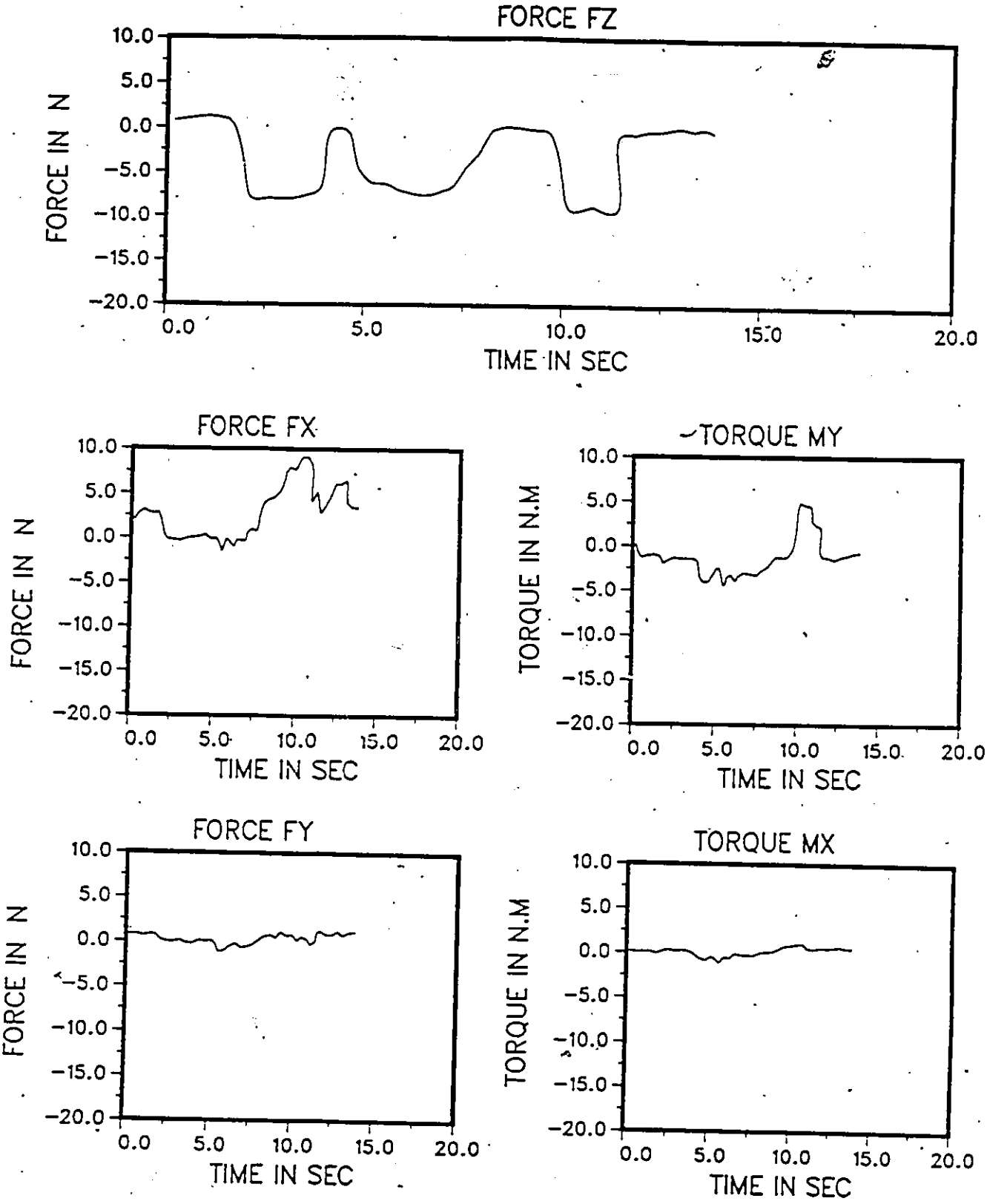


Figure 5.8: Experimental results, insertion forces for C = 0.002

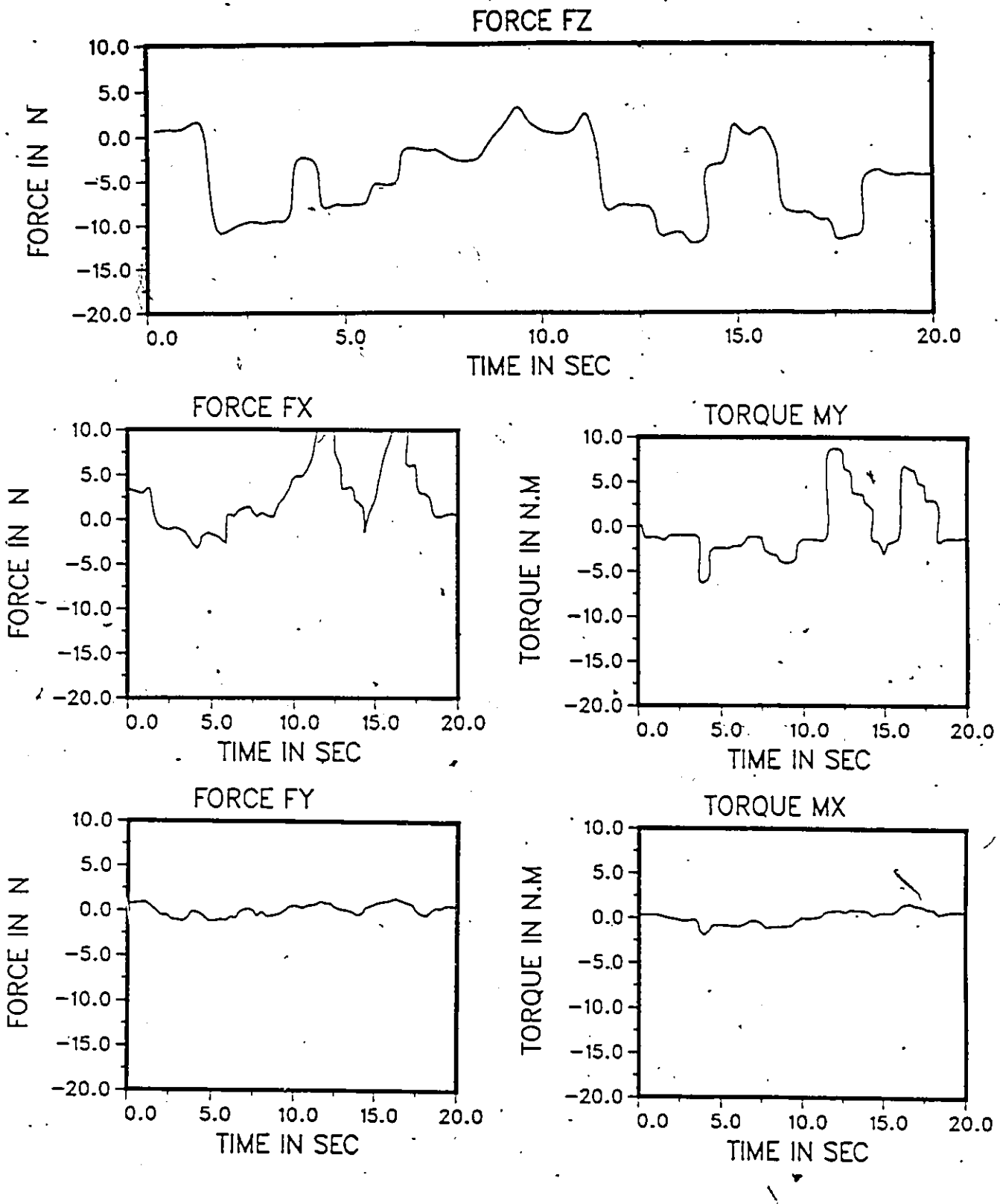


Figure 5.9: Experimental results, insertion forces for C = 0.001

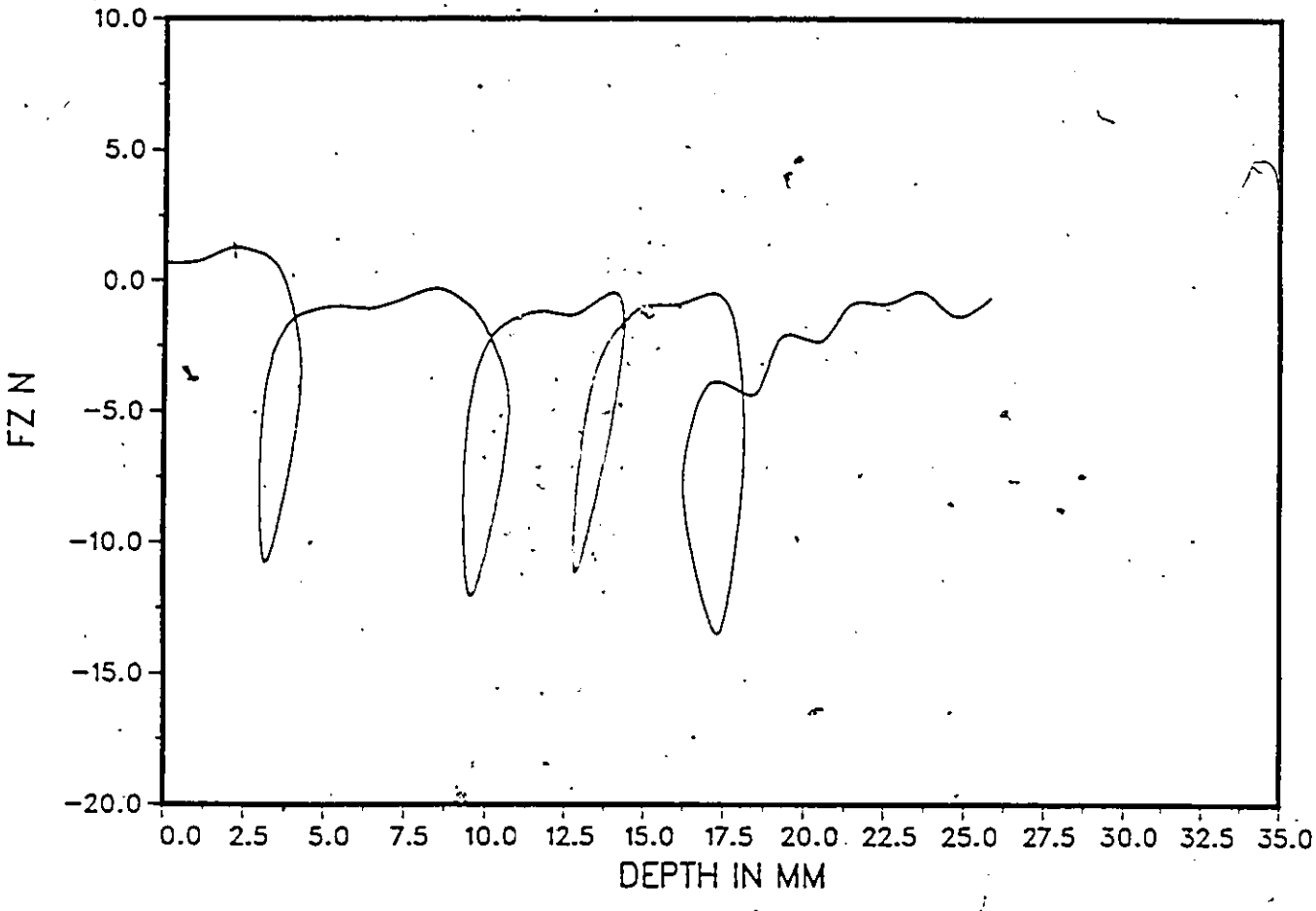


Figure 5.10: Experimental results, multiple wedging C = 0.001

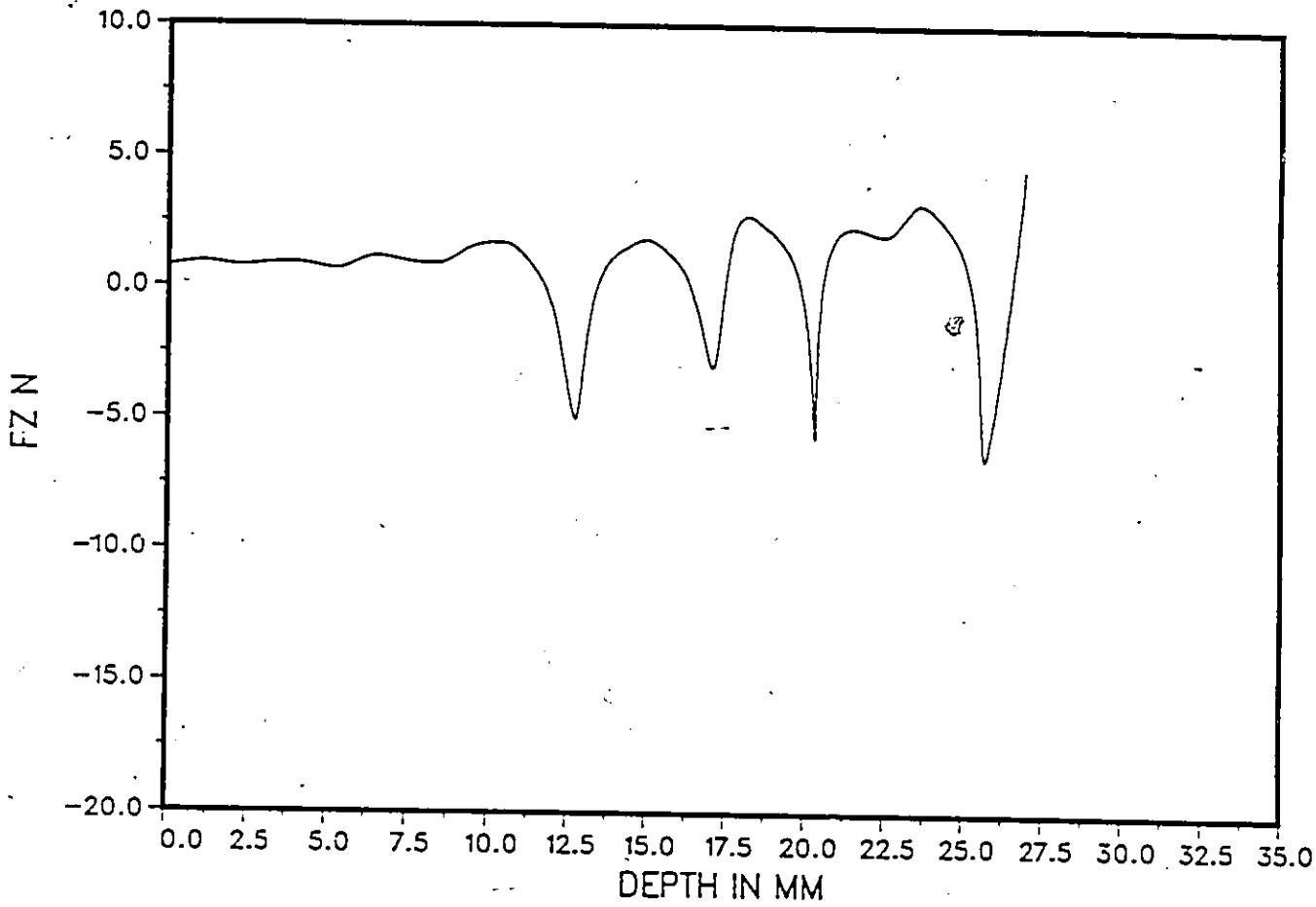


Figure 5.11: Experimental results, no wedging C = 0.002

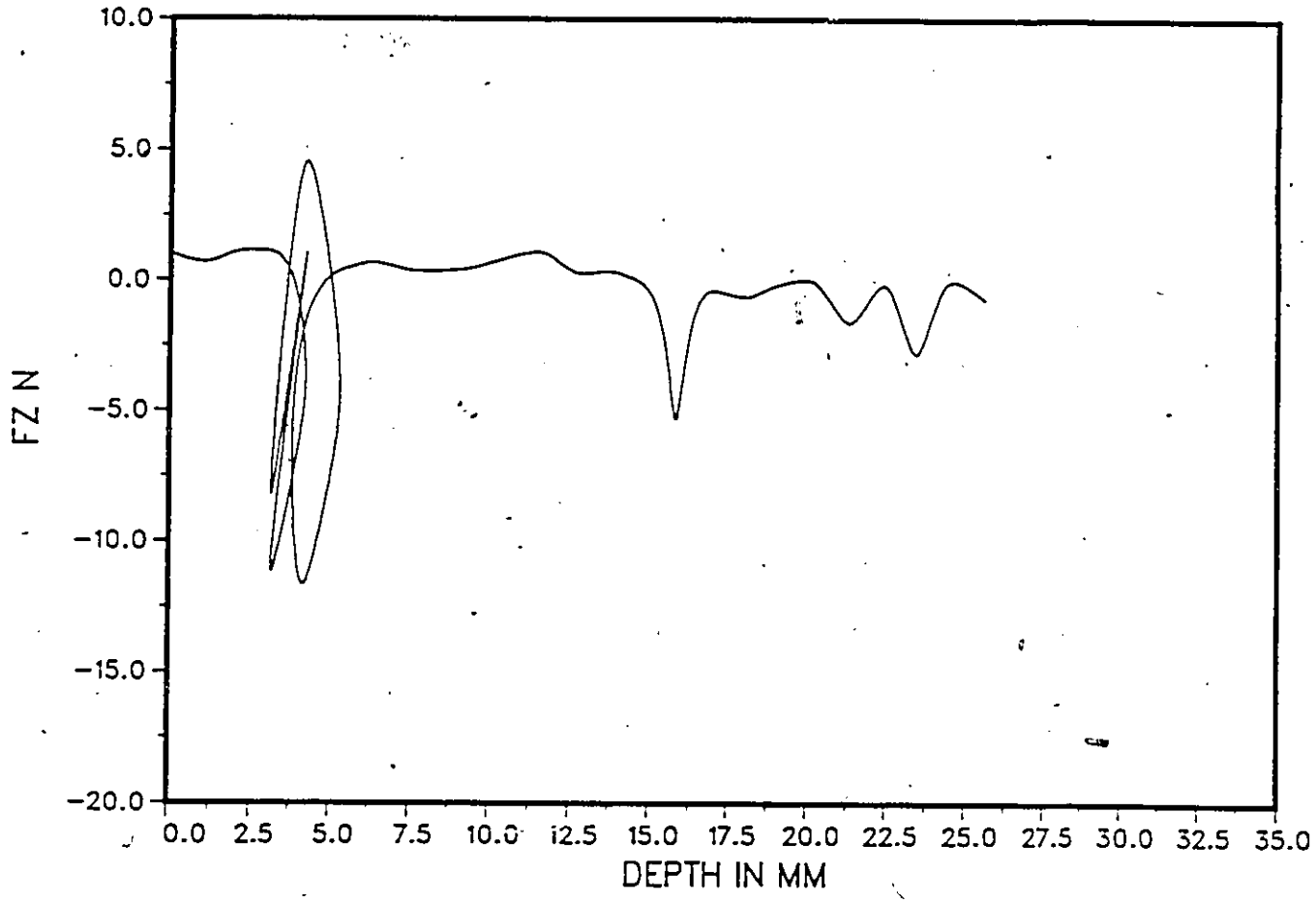


Figure 5.12: Experimental results, severe wedging C = 0.002

Chapter 6

Interaction of the Assembling Processor with the Rest of the System

As mentioned in chapter 2, the assembling processor is a distinct functional unit carrying out the task decomposition function and the object assembly sub-task. The interface signals are shown in Figure 6.1. Also, shown in Figure 6.1 are the physical object flow and the interaction of forces which occurs during the mating phase. After accepting an assignment as specified by the current task level instruction, the assembling processor issues the REQ(OBJNAME) signal requesting the delivery of the current object by the object presentation system which will deliver the requested object and

using its positional sensors will provide the absolute position, POSOBJ, of this object. This positional information is transmitted to the assembling processor as the parameter of the GNT(POSOBJ) granting signal. Upon receiving this signal, the assembling processor will be able to grasp the object delivered by object presentation system. After grasping the object, the assembling processor will carry out the "orientation & relative positioning" and the "mating" phases of the assembly task. The ROBSTATE communication line insures the synchronization between the task scheduler and assembling processor. This synchronization line take one of the following states:

- ASMFREE, which indicates that the assembling processor is free to accept a task from the task scheduler.
- ASMBUSY, which indicates that the assembling processor is currently busy executing an assembly task and is not able to accept a new task.
- OBJFAIL, which indicates that the assembling processor has failed in carrying out an assembly task. Being aware of this situation the task scheduler will reschedul the task. The assembling processor will consider that a task has failed if the robot does not grasp the object present at the end of the conveyor belt.
- MATFAIL, which indicates that the assembling processor failed to mate the object within a given amount of time, TOUT.

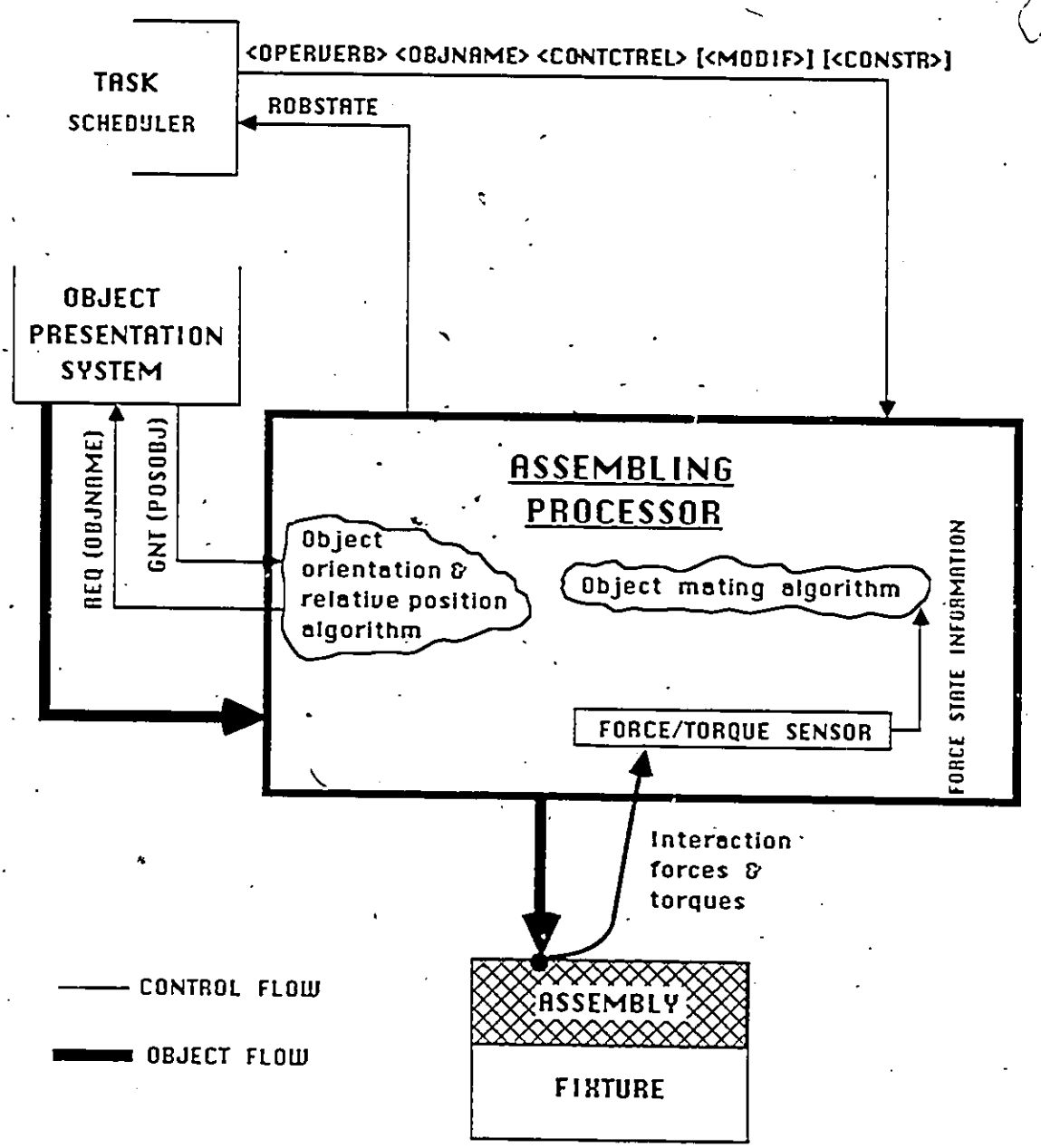


Figure 6.1: Interface of the assembling processor

Petri Net Model

In order to allows for a further performance evaluation of whole robotic assembly system we have opted for a Petri net modeling. The control of our assembly station is mainly concerned with the control signal flow and resource management. As a consequence, the Petri nets are particularly well suited for modeling. Petri net models are suitable to perform general validations such as liveness, boudedness, and absence of deadlocks, as well as to discover specific properties in the form of invariants. An important extension to Petri nets is the possibility of representing the time involved in the events which are modelled. Timed Petri nets have been used for the performance evaluation of concurrent systems such as communication protocols and multiprocessor computer systems. More recently, Petri nets are used for the modeling of manufacturing system with promising results [8].

The functional specification of our station is represented by the generalized stochastic Petri net sub-model given in Figure 6.1. Such stochastic Petri net is obtained by associating an exponentially distributed firing time with each transition of a Petri net. The generalized stochastic Petri nets includes both timed and immediate transitions. They are isomorphic to continuous time Markov chain, thus allowing the performance evaluation of the model to be carried out by classical probabilistic analysis. The diagram is fairly self-explanatory, however some comments are necessary. This model uses three types of transitions: immediate (boldline), determin-

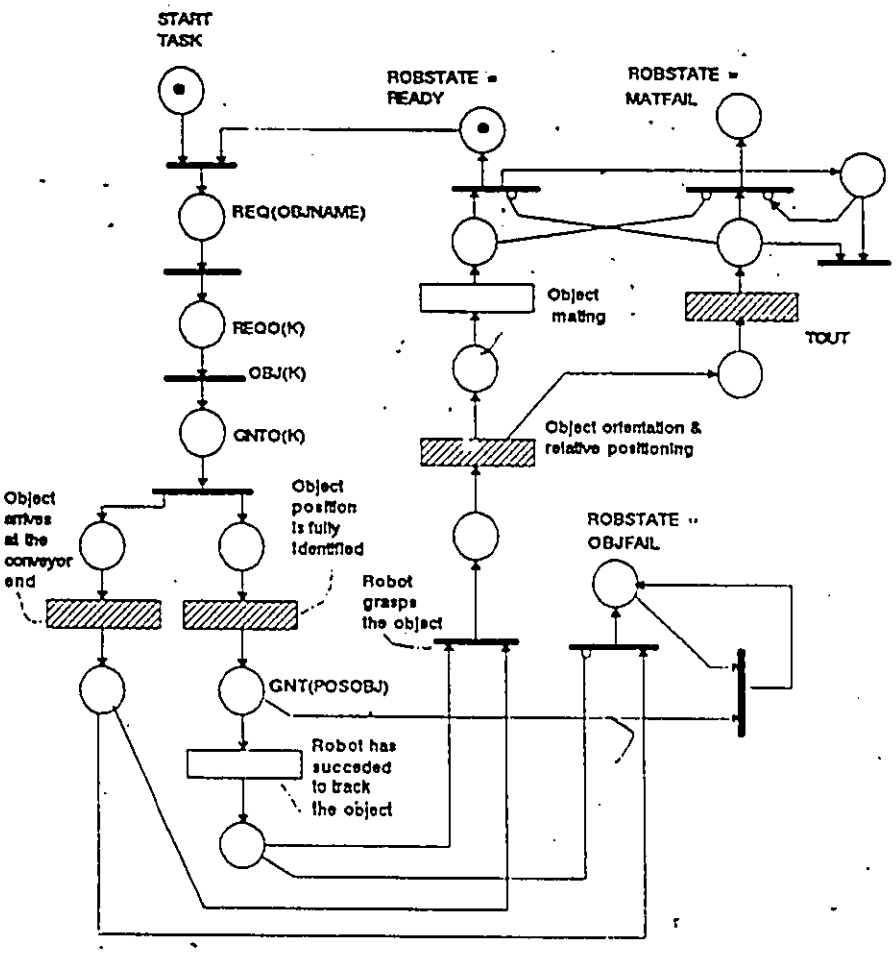


Figure 6.2: Petri nets modeling

istically timed (cross-hatched box), and stochastically timed (empty box). Tracking and grasping and mating are considered herein as being stochasticaly timed transitions. If the object arrives at the end of the conveyor and the robot fails to grasp it, the ROBSTATE communication line is then set to OBJFAIL indicating this failure to the TASK SCHEDULER. If the object mating does not succeed in the time allotted, TOUT, this is also considered a failure and ROBSTATE is set to the MATFAIL value.

Chapter 7

Conclusions

This work is part of a robotic assembling system conceived as a test bed for further research in task-level robotic programming language. The architecture of the robotic assembly system was implemented according to the structure of the task planner. Two distinct sub-tasks can be identified: "part fetching", and "part assembling". Corresponding to this partition the hardware architecture has two main distinct functional units: The "object presentation system" and the "assembling processor".

The object of this thesis was the development of the special functional unit executing the "part assembling" sub-task using "off the shelf" components, force sensor, robot arm, personal computer.

While the main goal of this work was the development of a functional

unit as part of a bigger system, it was necessary to implement position and force control algorithms for the specific off-the-shelf equipment presently used.

Based on the forward and inverse kinematics model of the SIR-1 robot, a position control algorithm is implemented. This position control is a point-to-point type of control. The precision obtained is $\pm 1mm$. No further improvement can be achieved without changing the mechanical and electrical characteristics of the arm. The accuracy obtained is limited by many factors such as the arm drive system (dc servo-motors and the optical encoders), the power transmission system, etc.

Using the ASTEK FS6-120A-100 force and torque sensor in the feedback loop, a closed loop force control algorithm is designed and implemented. This active force control is a cartesian based control and it is based on the following idea: the robot moves in a direction to minimize the reaction forces. The peg-in-hole operation was used as it was necessary to implement position This choice is justified by the frequent use of this task in assembly of manufacturing goods.

From this work the following major conclusion can be drawn:

- The partition of the robotics assembly system in functional units following the main partition lines of the task planner leads to a reduced complexity of the interface between the different parts of the system.

- Once the interfacing of the two functional units were defined, the design effort was concentrated on the problems related to that unit only. In our case the design of the ASSEMBLING PROCESSOR unit has focused on the integration of the three "off the shelf" components: robot system, force/torque sensor, and a personal computer acting as the local unit for the assembling processor. This integration is done by programs solving components intercommunication problems and implementing the position and force control algorithms required by the execution of the "object assembling" subtask. A parallel may be drawn between this approach and the microprogramming used in the development of computer system. Consequently, it becomes possible to extrapolate the classical microprogramming techniques for the design of our assembling processor.
- The C programming language adds to the control system many advantages such as portability and speed of execution. The use of high level language for the control of the assembling processor provides flexibility and makes it open for further integration within the structured assembly cell.

The experiments that we carried out to validate the functional concept lead to the following conclusions regarding the performance of the assembling processor:

- An active control method utilizing a sensor-based feedback system can handle the positioning error during the mating phase.

- The accidental passive compliance feature found in the robot arm seems to bring some contributions to the success of the mating phase but represents a drawback for the object orientation and positioning phase.

- As far as precision between the parts to be assembled is considered the results obtained are satisfactory. Parts having a clearance of up to 0.001 can be assembled.

- A relatively long insertion time is required due to the search motion time and, mainly, the limitation of the robot arm used in this work. Some of this limitation are:

  - the 5 degrees of freedom are insufficient for complete positioning and orientation of the robot.
  - The low accuracy and repeatability of the arm result in substantial positioning errors which are translated into the jamming of parts and the wedging situation.
  - While only point-to-point control can be achieved, the nature of the assembly task requires a more sophisticated path control method.

Future work includes the integration of the "assembling processor" into the rest of the robotic assembly cell. This assembly cell will represent the test bed for the task level programming.

While recognizing the limitation of this work, it is hoped that future researchers have been provided with more practical understanding of

- this complex research issue.

References and Bibliography

[1] J.F. Laszcz, *Product design for robotic and automatic assembly*, Proc. Robots 8 conference, Detoit, june 1984.

[2] R.H. Taylor and D.D. Grossman, *An integrated System Architecture*, Proc. of IEEE, vol.71, No. 7, pp. 842-856, july 1983.

[3] A.C. Sanderson and G. Perry, *Sensor-based robotic assembly systems: Research and application in electronic manufacturing*, Proc. of IEEE, vol.71, No. 7, pp. 857-871, july 1983.

[4] E. Petriu, T. Goguen and B. Karoui, *Multi-sensor robotic tracking and grasping*, ISA-DIGICON'87, Montreal, october 1987.

[5] T.L. Perez, *Robot programming*, Proc. of IEEE, vol. 71, No. 7 pp. 821-841, july 1983.

[6] R.C. Gonzalez, et al, *Robotics: Control, Sensing, Vision, and Intelligence*, McGraw-Hill, Inc, Newyork 1987.

[7] E. Petriu, B. Karoui, et al, *Distributed Robotic-assembly Architecture for a task-level programming language*, to be presented at MAPL symposium on manufacturing application language, June 1988.

[8] G. Bruno and P. Biglia, *Performance Evaluation and validation of tool handling in flexible manufacturing systems using Petri nets*, 1985 International workshop on timed Petri nets, Torino, Italy.

[9] SIR-1 user manuel, Scien-tech Intraco Automation (Pte) Ltd.

[10] T.L. Perez, *Compliance in robot manipulation*, Artificial Inteligence, pp. 5-12, 25(1985).

[11] B. Shimano and B. Roth, *On force sensing information and its use in controlling manipulators*, Proc. 9th international symposium on industial robot, pp. 119-128, Washington,D.C 1979.

[12] P.C. Watson and S.H. Drake, *Pedestral and wrist force sensors for automatic assembly*, Proc. of the 5th international symposium on industrial robot, pp. 501-511, september 1975.

[13] G. Piller, *A compact six-degree-of-freedom force sensor for assembly robot*, Proc. of the 12th International symposium on industrial robots, Paris, pp. 121-129, June 1982.

[14] FS6-120A 6-axis force/torque sensor software manuel, Astek engineering Inc, watertown, MA, 1986.

[15] J.L. Nevins and D.E. Whitney, *Computer-controlled assembly*, Scientific American, pp. 62-74, february 1978.

[16] B. Karoui and E. Petriu, *Contrôle de forces en assemblage avec bras de robot industriel*, 55^e congrès de l'ACFAS, Ottawa, mai 1987.

[17] D.E. Whitney, *Compliantly Supported Rigid Parts*, Journal of Dynamic System, Measurement and Control, March 1982.

[18] D.E. Whitney and J.L. Nevins, *What is the remote centre compliance (RCC) and what can it do ?*, Proc. 9th symposium on industrial robots, SME, Dearborn, MI, march 1979.

[19] T. Kasvand, *The state of robotics - A brief description*, National research council report.

[20] D.E. Whitney, *Historical Perspective and State of the Art in Robot Force Control*, The international journal of Robotics Research. Vol.6, No.1, Spring 1987.

[21] H. Asada and H. Yamamoto, *Torque feedback control of MIT direct-drive robot*, Proc. of the 14th international symposium on industrial robot, pp. 663-670, Gothenburg, Sweden, october 1984.

[22] R. Paul and B.E. Shimano, *Compliance and Control*, The 1976 joint automatic control conference, pp. 694-699, Albuquerque, NM, December 1976.

[23] J.K. Salisbury, *Active Stiffness Control of a Manipulator in Cartesian Coordinates*, Proc. 19th IEEE conference on decision and control, pp. 95-100, 1980.

[24] M.T. Mason, *Automatic planning of fine motion: correctness and completeness*, IEEE international conference on robotics, Atlanta, march 1984.

[25] M.T. Mason, *Compliant Motion*, in Robot Motion, Bradly et al (Eds), MIT Press, pp. 305-322, Cambridge, MA, 1983.

[26] M.T. Mason, *Compliance and force control for controlled manipulators*, IEEE transaction on system, man, and cybernetic, vol. SMC-11, No.6, pp. 418-432, june 1981.

[27] J.J. Craig, *Introduction to robotics mechanics & control*, Addison-Wesley publishing company, Inc. USA, 1986.

[28] M.H. Raibert and J.J. Craig, *Hybrid Position/Force Control of manipulator*, ASME journal of Dynamic, System, Measurement, Vol.102, pp. 126-133, June 1981.

[29] M.R. Cutkosky amd P.K. Wright, *Active control of compliant wrist in manufacturing tasks*, Proc. 14th International symposium on industrial robot, pp. 517-528, Sweden, october 1984.

[30] T.L. Defazio, et al, *The instrumented remote center compliance*, The industial robot, pp. 238-242, december 1984.

[31] M.R. Cutkosky and P.K. Wright, *Position sensing wrist for industrial manipulators*, Proc. of the 12th International symposium on industrial robots, pp. 427-438, Paris, June 1982.

[32] H.V. Brussel and J. Simons, *Automatic assembly by active force feedback accommodation*, Proc. the 8th international symposium on industrial robots, pp. 181-193, Stuttgart, west germany, 1978.

[33] Richard P. Paul, *Robot Manipulators: Mathematics, Programming, and Control*, The MIT Press, Cambridge, Massachusetts, 1981.

[34] Wesley E. Snyder, *Industrial Robots: Computer Interfacing and Control*, Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[35] Chi-Haur wu and R. Paul, *Resolved Motion Force Control of Robot Manipulator*, IEEE Transaction on System, Man and Cybernetics, Vol. SMC-12,No3, pp. 266-275, June 1982.

[36] D. Hill and R.J Vaccaro, *Cartesian Control of a robotic manipulator with joint compliance*, Robotica (1987) Vol.5, pp.207-215.

[37] Stepien T.M. et al, *Control of Tool/Workpiece contact force with application to robotic deburring*, Proc. IEEE conf. on Robotics and Automation, pp. 670-679, St. louis, 1985.

[38] R.k Roberts, et al, *The Effect of Force Sensor Stiffness on the Control of Robot manipulators*, Proc. IEEE conf. on Roboticcs and automation, pp. 269-274, St. louis, 1985.

[39] Tatsuo Goto, et al, *Control algorithm for precision insert operation robots*, IEEE Transaction on system, man and cybernetics, vol.SMC-10. No 1, pp. 19-25, January 1980.

[40] R.M. Inigo and R.M. Kossey, *Closed-loop control of a manipulator arm using a wrist force sensor*, IEEE transaction on industrial electronics, Vol. IE-34, No.3, pp. 371-378, August 1987.

[41] H.S. Cho, et al, *Robotic assembly: a synthesizing overview* Robotica (1987), volume 5, pp. 153-165.

[42] J. Spaa, et al, *Sensory controlled robot assembly and inspection: Hardware configuration*, Proc. of the 14th symposium on industrial robots, pp. 231-240, Cothenburg, Sweden, october 1984.

[43] B. Roth, et al, *On The theory of single and multiple insertions in industrial assembly*, Proc. of the 10th international symposium on industriel robots, pp. 545-558, Milan, march 1980.

[44] M.P. Hennesey, *Compliant part mating and minimum energy chanfer* 13th international symposium on industriel robots, pp. 79-92, Chicago, april 1983,

[45] R.N. Stauffer, *Robot in electronics assembly* Robotics today, pp. 61-67, december 1984.

[46] E.I Wood, *Robot in mechanical assembly* Robotics today, pp. 53-55, december 1985.

[47] D.E. Whitney, *Force feedback control of manipulator fine motions* journal of dynamic systems, measurement, and control, june 1977.

[48] J.P. Merlet, *A control law for the insertion of a flexible, cylindrical peg using a robot*, Proc. of the 3rd international conference on robot vision and sensory controls, pp. 453-460, Cambridge, November 1983,

[49] J.Berger, et al, *Robotic application in the assembly operations*, Proc. of the 12th international symposium on industrial robots, pp. 427-438, Paris, june 1982.

[50] A. Arnström and P. Gröndahl, *A flexible automatic assembly with automatic set-up*, Proc. of the 14th international symposium on industrial robot, pp. 397-406, Gothenburg, Sweden, october 1984.

[51] H.J. Warnecke, et al, *Programmable assembly cell for automotive parts and units*, Proc. '83 ICAR International Conference on Advanced Robotics, pp. 29-37, Japan, september 1983.

[52] T.L. De Fazio and D.E. Whitney, *Simplified Generation fo all Mechanical Assembly Sequences*, IEEE journal of robotics and automation, Vol.RA-3, No.6, December 1987.

[53] M. Kasai, et al, *Trainable assembly system with an active sensory table possessing six axes*, 11th international symposium on industrial robot, Tokyo, Japan, october 1981.

[54] A.J. Critchlow, *Introduction to Robotics*, Macmillan Publishing Company, New York, 1985.

[55] J.L. Peterson, *Petri Net Theory and Modeling of Systems*, Prentice-Hall Inc, Englewood Cliffs, N.J. 1981.

Appendix A

SIR-1 Command Set

SIR-1 has a set of host mode commands, these commands are sent from the host computer to the SIR-1 controller through the RS-232C serial interface. This enable the user to write their own programs to control the robot. The following are the set of these commands:

1- MOVE command

```
syntax: M  b, s, e, p, r, g, x, y  <cr>
  b, s, e, p, r, g, x, y represent the parameters for the
  joint motors, they are steps numbers.
  This command can activate the following:
     -move the motor(s) a specified number of steps;
     -output a patern through the output port x & y.
  It is not necessary to move all the 8 motors simu-
  ltaneously.
```

2- SPEED command

```
Syntax: S  b, s, e, p, r, g, x, y  <cr>
  b, s, e, p, r, g, x, y represent the parameters for the
  joint motors, they are speed numbers which range from 0
  to 7.
  Sir allows every motor to have its own moving speed.
```

3- PAUSE command

Syntax: P t <cr>
t is a number from 0 to 32767. This command cause the robot to pause for a 0.1xt seconds.

4- HOME command

Syntax: H <cr>
This command is used to bring the SIR-1 to its soft-home position.

5- TEACH command

Syntax: T <cr>
This command is used to activate the teach pendant during programming on the host computer.

6- READ command

Syntax: R B, S, E, P, R, G, X, Y <cr>
B, S, E, P, R, G, X, Y represent the parameters for the joint motors, their occurence is random. This command is used to examine the absolute position of various axes.

Appendix B

List of the sensor commands

OUTPUT

EO X Y <cr> Enable output

X = 1 ASCII output (default)
- Y = 1 Resolved forces and moments (default)
- Y = 2 LPF - BIAS
- Y = 3 LPF
- Y = 4 BIAS

X = 2 Packed output
- Y = 1 Forces and Moments (default)
- Y = 2 Forces only
- Y = 3 Moments only

X = 3 Analog output
- Y = 0 Zero volts (default)
- Y = 1-6 Mx-Mz
- Y = 7 Fx-Mz sequence
- Y = 11-16 LPF 1-6
- Y = 17 LPF sequence
- Y = 21 240 hz square wave, full scale
- Y = 22 1 hz square wave, full scale
- Y = 23 1 hz triangle, full scale
- Y = 24 Small sawtooth
- Y = 25 + full scale, DC

Y = 26 - full scale, DC

X = 4 12-byte format.

BIAS

RB <cr> Reset Bias

FRAME

CF X Y <cr> Change Frame
X = 1 - 3 (X,Y,Z)
Y = 0 - 500 mm

DF <cr> Disply Frame

RF <cr> Reset Frame

LIMITS

CL X Y <cr> Change limits
X = 1 - 6 (Fx-Mz)
Y = New Value

DL <cr> Display limits

RL <cr> Reset limits

IL <cr> Inhibit limits

PASSBAND

CP X <cr> Change Passband
X = 1 240 hz (bypass)
X = 2 120 hz
X = 3 60 hz
X = 4 30 hz
x = 5 15 hz

DP <cr> Display Passband

THRESHOLD

CT X <cr> Change Threshold

X = 0 - 4095 New threshold

DT <cr> Display Threshold

RT <cr> Reset Threshold

IT <cr> Inhibit Threshold

RATE

CR X <cr> Change Rate

X = 1	38400 baud
X = 2	19200 baud
X = 3	9600 baud
X = 4	4800 baud
X = 5	2400 baud
X = 6	1200 baud
X = 7	600 baud
X = 8	300 baud

DR <cr> Display Rate

Appendix C

SIR-1 Utility Library

```
/*
 *                    SIR.H LIBRARY
 *
 * This library contains all the functions needed to operate the
 * the robot. Two sets of function can be found. The first set
is
 * a collection of function that implement the robot commands set
 * and the second are the function that implement the forward and
 * inverse kinematics of the sir-1 robot arm.
 */

#define home  send(H,com2);cari_ret(com2)

      /* Global variables */
int *ptrp;        /* number of steps of each motor. */
double *point1;   /*  cartesian coordinates (forward solu.) */
double *point2;   /*  angular coordinate (forward solution) */
double angle[5]; /* values of the angles (inverse solu.) */

/*
 * This function initialize the communiction between the robot
 * controller and the computer
 */
void  open_com1()
 {
   int  ch;
   cominit2();
```

```
   send(escape,com2);
   cari_ret(com2);
   if(read_com(com2) <= 0)
   {printf("Set the robot controller to the host mode & try again
\n");
   abort();}
   printf("THE SIR-1 ROBOT IS SET\n");
 }

/*
 * Function to convert a string of cartacter to an integer
 */
int near car_int(char ch[])
 {
    int posit;
    char  *iptr;
    iptr=ch;
    posit=atoi(iptr);
    return(posit);
  }

/*
 * This function converts an integer to a string of carracter
 * it return a pointer to the string
 */
char near *int_car(int valu)
  {
    char *iptr,string[7];
    iptr=string;
    ltoa(valu,iptr,10);
    return(iptr);
  }

/*
 * This function counts the number of carracter in an integer
 */
int near  count(int value)
  {
    int max;
    if(value<10 && value>0)  max=0;
    else if((value<100 && value>0)||(value>-10 && value<0))    max=1;
    else if((value<1000 && value >0) ||(value>-100 && value<0))
max=2;
    else if((value<10000 && value<0)||(value>-1000 && value<0))
max=3;
    else max=4;
    return(max);
  }

/*
 * This function returns the present reading of the optical encoders.
```

```
 * The different values are contained in a pointer called PTRP.
 */
void near act_posit()
{
  int i,j,step[7];
  char ch,ch1[7],ch2[7],ch3[7],ch4[7],ch5[7],ch6[7],ch7[7],ch8[7],ch9[4];
  ptrp=step;
    send(R,com2);send(space,com2);
    send('B',com2);send(comma,com2);
    send('S',com2);send(comma,com2);
    send('E',com2);send(comma,com2);
    send('P',com2);send(comma,com2);
    send('R',com2);send(comma,com2);
    send('G',com2);cari_ret(com2);
    ch=check();
    if(ch=='?')
      {
        printf("%c\n",ch);
        j=0;
      }
    else
      {
        ch1[0]=ch;
        j=1;
      }
    for(i=j;i<=6;i++)  ch1[i]=check();
    for(i=0;i<=6;i++)  ch2[i]=check();
    for(i=0;i<=6;i++)  ch3[i]=check();
    for(i=0;i<=6;i++)  ch4[i]=check();
    for(i=0;i<=6;i++)  ch5[i]=check();
    for(i=0;i<=6;i++)  ch6[i]=check();
    step[0]=car_int(ch1);step[1]=car_int(ch2);
    step[2]=car_int(ch3);step[3]=car_int(ch4);
    step[4]=car_int(ch5);step[5]=car_int(ch6);
         /*
          *   BASE         : step[0]
          *   SHOULDER     : step[1]
          *   ELBOW        : step[2]
          *   WRIST PITCH  : step[3]
          *   WRIST ROLL   : step[4]
          *   GRIPPER      : step[5]
          */
    }

/*
 * This function return the present position of the robot arm.
 * POINT1 contains the cartesian position.
 * POINT2 contains the angular position.
 */
```

```
void near position()
  {
    int i,j,posit,step[7];
    double O1,O2,O3,O4,O5,X,Y,Z,x,z,a2,a3,a4,PI;
    double ny,oy,ax,az,gamma,beta;
    double coordinate[5],angle[5];
    point1=coordinate;
    point2=angle;
    a2=204.0;a3=250.0;a4=0.0;PI=pi;
    act_posit();
    for(i=0;i<=4;i++)
      step[i]=*(ptrp+i);
    O1=step[0]*0.24*PI/180;
    O2=step[1]*0.03*2*PI/360-PI/2;
    O3=step[2]*0.023*2*PI/360-O2;
    O4=step[3]*0.18*PI/180;
    O5=step[4]*0.9*PI/180;
    ny=sin(O1)*cos(O2+O3+O4)*cos(O5)+cos(O1)*sin(O5);
    oy=-cos(O2+O3+O4)*sin(O5)*sin(O1)+cos(O1)*cos(O5);
    ax=cos(O1)*sin(O2+O3+O4);
    az=cos(O2+O3+O4);
    X=cos(O1)*(cos(O2+O3)*a3+cos(O2)*a2);
    Y=sin(O1)*(cos(O2+O3)*a3+cos(O2)*a2);
    Z=-sin(O2+O3)*a3-sin(O2)*a2;
    gamma=atan2(ny,oy);
    beta=atan2(ax,az);
    coordinate[0]=X;
    coordinate[1]=Y;
    coordinate[2]=Z;
    coordinate[3]=gamma*180/PI;
    coordinate[4]=beta*180/PI;
    angle[0]=O1;
    angle[1]=O2+PI/2;
    angle[2]=O3+O2;
    angle[3]=O4;
    angle[4]=O5;
  }

/*
 * This function takes an integer and sended throught com2
 */
 void near send_int(int value)
  {
    int  j,max,inter;
    char  *ptr,chr[5];
    max=count(value);
    ptr=int_car(value);
```

```
    chr[0]=*ptr;chr[1]=*(ptr+1);chr[2]=*(ptr+2);
    chr[3]=*(ptr+3);chr[4]=*(ptr+4);
    for(j=0;j<=max;j++)
      {
        inter=chr[j];
        send(inter,com2);
      }
  }

/*
 * This function controls the speed of the robot arm.
 */
void near speed(int speed)
  {
    int i;
    char *spee;
    spee=int_car(speed)
    send('S',com2);
    send(space,com2);
    for ( i=0;i<=4;i++ )
     {
       send(*spee,com2);
       send(comma,com2);
     }
    send(*spee,com2);
    cari_ret(com2);
  }

\*
 * This function takes a pointer that contains the number of steps
 * for each joint and sends them to the robot controler.
 */
void near move(int *ptry)
  {
    int i,posit[6];
    posit[0]=*ptry;posit[1]=*(ptry+1);posit[2]=*(ptry+2);
    posit[3]=*(ptry+3);posit[4]=*(ptry+4);posit[5]=*(ptry+5);
    send(M,com2);
    send(space,com2);
    for(i=0;i<=5;i++)
      {
        if(posit[i] !=0)
        send_int(posit[i]);
        if(i<5)
          send(comma,com2);
      }
    cari_ret(com2);
  }
```

```
void near move_joint(int *ptry)
  {
    int i,posit[6],*ptr;
    ptr=posit;
    posit[0]=*ptry;posit[1]=*(ptry+1);posit[2]=*(ptry+2);
    posit[3]=*(ptry+3);posit[4]=*(ptry+4);posit[5]=*(ptry+5);
    for(i=0;i<=5;i++)
    move(ptr);
  }

/*
 * This function makes the robot pause for an amount of time given
 * by "time".
 */
void near pause(int time)
  {
    send(P,com2);
    send(space,com2);
    send_int(time);
    cari_ret(1);
  }

/*
 * When one or more joint values are out of range, this function
 * print a error message along with the corresponding joint.
 */
void near error(int ch)
  {
    if(ch <= 6 )
      {
        printf("VALUE OUT OF RANGE ");
        switch(ch)
          {
            case 1 :printf("'BASE'\n");
                    break;
            case 2 :printf("'SHOULDER'\n");
                    break;
            case 3 :printf("'ELBOW'\n");
                    break;
            case 4 :printf("'WRIST PITCH'\n");
                    break;
            case 5 :printf("'WRIST ROLL'\n");
                    break;
            case 6 :printf("'GRIPPER'\n");
                    break;
          }
        printf("CHECK YOUR ENTRIES\n");
      }
  }
```

```
/*
 * This function takes the controller return value after initialization
 * and gives the status of the different moters.
 */
void near diagnostic(int ch)
  {
    printf("THE WORKING MOTORS ARE :\n");
    if ( ch >= 128 )
      { printf("   Y motor\n"); ch=ch-128; }
    if ( ch >= 64 )
      { printf("    X motor\n"); ch=ch-64; }
    if ( ch >= 32 )
      { printf("    GRIPPER\n"); ch=ch-32; }
    if ( ch >= 16 )
      { printf("    WRIST ROLL\n"); ch=ch-16; }
    if ( ch >= 8 )
      { printf("    WRIST PITCH\n"); ch=ch-8; }
    if ( ch >= 4 )
      { printf("    ELBOW\n"); ch=ch-4; }
    if ( ch >= 2 )
      { printf("    SHOULDER\n"); ch=ch-2; }
    if ( ch >= 1 )
      { printf("    BASE\n"); }
    if ( ch == 0 )
      { printf ("   NONE\n"); }
}
/*
 * Given the cartesian coordinates this function computes the
 * different joints angles. The angles are stored in an array
 * called ANGLE[].
 */
void near comp_angle(double px,double py,double pz,double B1,double
B2)
  {
    double  a2,a3,a4,ax,ay,az,nx,ny,ox,oy,valu1,valu2;
    double O1,O2,O3,O4,O5,O123,PI,W[5];
    PI=pi;
    a2=204.0;a3=250.0;
    ax=sin(B2)*cos(B1);
    ay=sin(B1)*sin(B2);
    az=cos(B2);
    nx=cos(B2)*cos(B1);
    ny=sin(B1)*cos(B2);
    ox=sin(B1);
    oy=cos(B1);
    O1=atan2(py,px);
    O5=atan2(-nx*sin(O1)+ny*cos(O1),-sin(O1)*ox+oy*cos(O1));
    O123=atan2(ax*cos(O1)+ay*sin(O1),az);
```

```
    valu1=cos(O1)*px+sin(O1)*py;
    valu2=(valu1*valu1+pz*pz-a3*a3-a2*a2)/(2*a3*a2);
    if(valu2==0)
    O3=PI/2;
    else  O3=atan2(sqrt(1-valu2*valu2),valu2);
    O2=-atan2((a3*cos(O3)+a2)*pz-a3*sin(O3)*valu1,(a3*cos(O3)+
    a2)*valu1+a3*sin(O3)*pz);
    O4=O123-O2-O3;
    position();
    W[0]=*point2;      W[1]=*(point2+1);
    W[2]=*(point2+2); W[3]=*(point2+3);
    W[4]=*(point2+4);
    angle[0]=O1-W[0];
    angle[1]=O2-W[1];
    angle[2]=O3-W[2];
    angle[3]=O4-W[3];
    angle[4]=O5-W[4];
}
```

Appendix D

User Programs

```
/*
 * This program position the robot in the cartesian space.
 * It takes, as parameter, the coordinate of the desired
 * position.
 */
#pragma inline
#include<dos.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include"const.h"
#include"asm.h"
#include"sir.h"

main(argc,argv)
 int argc;
 char *argv[];
  {
    int  i,j,n,posit[6],*point_pos,flag1,pos[7],grip,B1,B2,cart[6];
    double PI,BT1,BT2,arra[6],posi[5],ang[5],temp;

    PI=pi;
    point_pos=posit;
    if(argc < 6)
     printf("missing %d parameters\n",7-argc);
    else if(argc >6)
     printf("more parameters than needed\n");
```

```
else
{
  for(i=1;i<=5;i++)  cart[i-1]=posi_value(argv[i]);
  open_com1();
  for(n=0;n<=1;n++)
   {
     position();
     for(i=0;i<=5;i++) pos[i]=*(ptrp+i);
     for(j=0;j<=4;j++) arra[j]=(double) cart[j];
     B1=arra[3];B2=arra[4];temp=arra[2];
     BT1=B1*PI/180;BT2=B2*PI/180;
     if(n==0) arra[2]=*(point1+2);
     else if(n==1) arra[2]=temp+20.0;
     else arra[2]=temp;
     comp_angle(arra[0],arra[1],arra[2],BT1,BT2);
     posi[0]=(angle[0]*180.0)/(PI*0.24);
     posi[1]=(angle[1]*180.0)/(PI*0.03);
     posi[2]=(angle[2]*180.0)/(PI*0.02);
     posi[3]=(angle[3]*180.0)/(PI*0.18);
     posi[4]=(angle[4]*180.0)/(PI*0.9);
     for(i=0;i<=4;i++) posit[i]=(int) (posi[i]+0.5);
     posit[5]=0;
     if((posit[0]+pos[0])>600 || (posit[0]+pos[0])<-600)
       { error(1); flag1=0;}
     if((posit[1]+pos[1])>1200 || (posit[1]+pos[1])<-910)
       { error(2); flag1=0;}
     if((posit[2]+pos[2])>1270 || (posit[2]+pos[2])<-810)
       { error(3); flag1=0;}
     if((posit[3]+pos[3])>630 || (posit[3]+pos[3])<-280)
       { error(4); flag1=0;}
     if((posit[4]+pos[4])>490 || (posit[4]+pos[4])<-280)
       { error(5); flag1=0;}
     if(flag1 == 0)
       abort();
     else
       move_joint(point_pos);
   }
   position();
   printf("\n X= %f  Y= %f  Z= %f \n",*point1,
   *(point1+1),*(point1+2));
   printf("roll= %f degrees  pitch= %f degrees
   \n\n",*(point1+3),*(point1+4)      );
   exit(0);
 }
}
/*
```

```
 * This program controls the opening of the griper.
 * It takes the number of steps for the griper motor
 * as a parameter.
 */
#pragma inline
#include<stdio.h>
#include<stdlib.h>
#include"const.h"
#include"asm.h"
#include"sir.h"

main(argc,argv)
 int argc;
 char *argv[];
 {
  int i,j,*res_p;
  if(argc <2) printf("missing the parameter of the griper\n");
  else if(argc >2) printf("extra parameters\n");
  else
    {
      res_p=(int *) malloc(6*sizeof(int));
      act_posit();
      for(j=0;j<=5;++j)
      *(res_p+j)=0;
      *(res_p+5)=posi_value(argv[1])-*(ptrp+5);
      move(res_p);
      act_posit();
      free(res_p);
      exit(0);
    }
 }
```

Appendix E

I/O Librairy

```
/*
 *                    I/O.H
 *
 * This is the I/O librairy. The code is written using the 86/88
 * assembly language. This functions can be accessed by simple
 * calls. Both the robot and the sensor programs are using this
 * librairy.
 */

/*
 * This function sends a charracter specified by "message" to
 * either one of the serial ports "com0".
 */
void near send (int message,int com0)
   {
        asm     mov     dx,com0
        asm     mov     ax,message
        asm     mov     ah,1
        asm     int     14h
   }

/*
 * This function initialize the RS-232 port (com1 ) to pe
 * used for the sensor.
 */
void near cominit1()
```

```
    asm     mov   al,10010011b /* 1200 baud,no parity,8 data bit
*/
    asm     mov   dx,com1
    asm     mov   ah,0
    asm     int   14h
    }

/*
 * This function initialize the RS-232 port ( com2 ) to be
 * used for the robot controller.
 */
void near cominit2()
  {
    asm    mov    al,11110011b /* 9600 baud,no parity,8 data bit
*/
    asm    mov    dx,com2
    asm    mov    ah,0
    asm    int    14h
  }

/*
 * This function checks the input line for a charracter. It returns
 * the carracter found or set the flag to zero and return garbage.
 */
int near check_inp()
  {
    asm    push   ds
    asm    push   dx
    asm    push   si
    asm    mov    si,dx
    asm    add    si,si
    asm    mov    dx,40h
    asm    mov    ds,dx
    asm    mov    dx,[si]
    asm    add    dx,5
    asm    in     al,dx
    asm    test   al,1
    asm    jz     exit
    asm    mov    dx,[si]
    asm    in     al,dx
exit:
    asm    pop    si
    asm    pop    dx
    asm    pop    ds
  return(_AX);
}

/*
 * This function sends a cariage-return to the serial port
 * specified by "com0".
 */
void near cari_ret(int com0)
   {
```

```
        send(13,com0);
        send(10,com0);
    }

/*
 * This function reads the serial port specified by "com"
 * and it returns the value read. If no value is present,
 * at the port, then after some tries a -1 is returned.
 */
int near  read_com(int com)
  {
   int i=0;
    asm    mov   dx,com
    a: i++;
    if(i>=50)
     return(-1);
    else
     {
       check_inp();
       asm    jz    a
       return(_AL);
     }
  }

/*
 *                  CONST.H
 *
 * This library contain some predefined constant used
 * by the different programs.
 */
#define cr  13
#define lf  10
#define A   65
#define E   69
#define O   79
#define R   82
#define B   66
#define I   73
#define C   67
#define F   70
#define D   68
#define L   76
#define P   80
#define T   84
#define H  72
#define M  77
#define pi 3.14
#define zero  48
#define one   49
#define two   50
#define three 51
#define four  52
#define five  53
#define six   54
#define space 32
#define comma  44
```

```
#define escape 27
#define com1   0
#define com2   1
```

Appendix F

Sensor Library

```
/*
 *                    SENSOR.H
 *
 * This is the interface library of the sensor. All the
 * commands of the sensor are implemented using the C
 * language. In addition to the command a set of function
 * to read the forces and the torques are implemented.
 */

/*
 * This function implement the IO command which is the
 * inhibit output command. Once this function is used the
 * sensor microprocessor stops sending the forces.
 */
void near inhibit_output()
   {
     int  ch;
        send(I,com1);
        send(O,com1);
        cari_ret(com1);
        ch=' ';
        while(ch != '?')
        ch=get_port();
    }

/*
 * This function resets the bias by sending the RB command
```

```
 * to the sensor microprocessor. This function has the effect
 * of zeroing the reading of the sensor regardless of the
 * sensor loading.
 */
void near reset_bias()
   {
     int  ch;
       send(R,com1);
       send(B,com1);
       cari_ret(com1);
       ch=' ';
       while(ch !='?')
       ch=get_port();
    }

/*
 * This is a counter that takes an integer and returns the
 * number of charracters it contains.
 */
int near  counter(int value)
  {
    int max;
    max=0;
  while(value != 0)
   {
    value=value/10;
    ++max;
   }
    return(max);
  }

/*
 * This function converts an array of charracters to an
 * integer.
 */
int near posi_valu(char ch[])
  {
    int posit;
    char  *iptr;
    iptr=ch;
    posit=atoi(iptr);
    return(posit);
 }

/*
 * This function takes an integer and sends it to the serial
 * port "com1" where the sensor is connected.
 */
void  near  send_in(int value)
 {
  int  j,max,inter;
  char  *ptr,chr[4];
```

```
  ptr=chr;
  max=counter(value);
  itoa(value,ptr,10);
  chr[0]=*ptr;chr[1]=*(ptr+1);chr[2]=*(ptr+2);
  for(j=0;j<=max;j++)
  {
    inter=chr[j];
    send(inter,com1);
  }
}
/*
 * This function reads a voltage ranging from 5V to -5V
 * through the analog/digital port and return this value
 * to the calling program.
 */
float near force()
 {
   float  f1;
   const K=2048;
   int  A0,A1,A2,A3;
   int i,j;
   for(j=1;j<3;j++)
   A1=inportb(0x304); /* read port 772 to start conversion */
   for(j=1;j<3;j++)
   A0=inportb(0x306); /* read the MSB at adress 774 */
   for(j=1;j<3;j++)
   A2=inportb(0x307); /* read the LSB at adress 775 */
   A3=A0*16+A2/16;    /* reconstruct the value */
   f1=(float)(A3);    /* convert to real value */
   return((f1-K)*5/K);/* return the value in voltage */
 }
/*
 * This function enable the output of the sensor. It takes
 * two parameters x & y. With x=3 the output is enabled for
 * analog transmission. y takes values from 1 to 6: 1 for Fx
 * 2 for Fy, 3 for Fz, 4 for Mx, 5 for My, and 6 for Mz.
 */
void near enable_output(x,y)
  int x,y;
   {
     int  ch;
       send(E,com1);
       send(0,com1);
       send(space,com1);
       send(x,com1);
       send(space,com1);
       send(y,com1);
```

```
        cari_ret(com1);
        ch=' ';
        while(ch !='?')
        ch=get_port();
    }
/*
 * This function returns the value of Fx in [N].
 */
float near Fx()
    {
     float ff;
     enable_output(three,one);
     ff=force();
     inhibit_output();
     return(ff*20);
    }
/*
 * This function returns the value of Fy in [N].
 */
float near Fy()
    {
     float ff;
     enable_output(three,two);
     ff=force();
     inhibit_output();
     return(ff*20);
    }
/*
 * This function returns the value of Fz in [N].
 */
float near Fz()
    {
     float ff;
     enable_output(three,three);
     ff=force();
     inhibit_output();
     return(ff*20);
    }
/*
 * This function returns the value of Mx in [N.M].
 */
float near Mx()
    {
     float ff;
     enable_output(three,four);
     ff=force();
     inhibit_output();
```

```
   return(ff*4);
  }
/*
 * This function returns the value of My in [N.M].
 */
float near My()
  {
   float ff;
   enable_output(three,five);
   ff=force();
   inhibit_output();
   return(ff*4);
  }
/*
 * This function returns the value of Mz in [N.M].
 */
float near Mz()
  {
   float ff;
   enable_output(three,six);
   ff=force();
   inhibit_output();
   return(ff*4);
  }
/*
 * This function initialize the communication link between
 * the sensor and the computer and resets the bias.
 */
void  near open_com_link()
 {
   int resul;
   char ch;
   cominit1();
   cari_ret(com1);
   if(read_com(com1) <= 0)
     {
       printf("turn on the sensor and try again\n");
       abort();
     }
   resul=' ';
   while(resul != one )
     resul=read_com(com1);
   while(resul != two)
       resul=read_com(com1);
   while(resul != zero )
       resul=get_port();
   while(resul != A)
       resul=get_port();
```

```
    printf("\nF S 6 - 1 2 0  A\n");
    printf("FORCE  SENSOR  IS  SET  \n\n");
  }
/*
 * This function changes the frame of the sensor in the
 * cartesian frame. x specifies the the axis: 1 for x-axis
 * 2 for y-axis, and 3 for z-axis. y [mm] specifies the
 * displacement along the specified axis.
 */
void near change_frame(int x,int y)
  {
      send(C,com1);
      send(F,com1);
      send(space,com1);
      send_in(x);
      send(space,com1);
      send_in(y);
      cari_ret(com1);
  }
/*
 * This function displays the present sensor frame.
 */
void near display_frame()
  {
   int    i;
   int  ch;
   ch=0;
      send(D,com1);
      send(F,com1);
      cari_ret(com1);
      printf("THE CURRENT LOCATION OF THE FRAME IS :\n");
      putc(ch);
      while(ch != '?')
      ch=get_port();
  }
/*
 * This function resets the frame to the origin of
 * the sensor.
 */
void near reset_frame()
  {
      send(R,com1);
      send(F,com1);
      cari_ret(com1);
  }
/*
 * This function allows any one of the 6 limit values to
```

```
 * be changed to a new value. For example, x=3 and y=200
 * would change the Z force to 200 [N].
 */
void near change_limits(int x,y)
   {
        send(C,com1);
        send(L,com1);
        send(space,com1);
        send_in(x);
        send(space,com1);
        send_in(y);
        cari_ret(com1);
  }

/*
 * This function displays the current settings of the
 * 6 limit value.
 */
void display_limits()
   {
     int   i;
     char  ch;
     send(D,com1);
     send(L,com1);
     cari_ret(com1);
     printf("THE CURRENT SETTINGS OF THE 6 LIMIT VALUES ARE :\n");
     for(i=1;i<100;i++)
       {
        ch=get_port();
        printf("%c",ch);
       }
  }

/*
 * This function resets the limit values to the default
 * values which are set at power-on.
 */
void near reset_limits()
   {
        send(R,com1);
        send(L,com1);
        cari_ret(com1);
  }

/*
 * This function sets the limits to their maximum values
 */
void near inhibit_limits()
   {
        send(I,com1);
        send(L,com1);
```

```
        cari_ret(com1);
  }
/*
 * This function displays the current selection of
 * the low pass filter.
 */
void display_passband()
   {
     int   i;
     char  ch;
     send(D,com1);
     send(P,com1);
     cari_ret(com1);
     printf("THE CURRENT LPF SELECTION IS :\n");
     for(i=1;i<10;i++)
       {
        ch=get_port();
        printf("%c",ch);
       }
  }
/*
 * This function changes the setting of the low pass
 * filter. x can takes values from 1 to 5 where:
 * 1 -> (240 hz), 2 -> (120 hz), 3 -> (60 hz),
 * 4 -> (30 hz), 5 -> (15 hz).
 */
void near change_passband(int x)
   {
        send(C,com1);
        send(P,com1);
        send(space,com1);
        send_in(x);
        cari_ret(com1);
  }
/*
 * This function displays the current excess rate threshold.
 */
void display_threshold()
   {
     int   i;
     char  ch;
     send(D,com1);
     send(T,com1);
     cari_ret(com1);
     printf("THE CURRENT SETTINGS OF THE EXCESS RATE THRESHOLD
:\n");
     for(i=1;i<10;i++)
       {
```

```
        ch=get_port();
        printf("%c",ch);
       }
  }

/*
 * This function changes the excess rate threshold
 * x is the new threshold and can varies from 0 to 4095.
 */
void near change_threshold(int x)
   {
        send(C,com1);
        send(T,com1);
        send(space,com1);
        send_in(x);
        cari_ret(com1);
  }

/*
 * this function resets the excess rate threshold to the
 * power-on default value.
 */
void near reset_threshold()
   {
        send(R,com1);
        send(T,com1);
        cari_ret(com1);
  }

/*
 * As soon as this function is executed the sensor will
 * switch to a new rate indicated by x. x can be a value
 * from 1 to 8 where 1 correspond to 38400 and 8 to 300 baud.
 */
void near change_rate(int x)
  {
      send(C,com1);
      send(R,com1);
      send(space,com1);
      send_in(x);
      cari_ret(com1);
  }
```

Appendix G

Peg-In-Hole

```
/*
 *                          .PEG_IN_HOLE
 *
 * This is the code of the peg-in-hole program. The peg is
 * assumed to be on top of the hole. The program will insert
 * the peg, display the time taken by this operation and then
 * open the gripper and moves the end-effector away from the
 * hole. On success a 0 is returned.
 */

#pragma inline
#include<process.h>
#include<dos.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include"const.h"
#include"asm.h"
#include"sensor.h"
#include"sir.h"

 int *resol_p[8]; /* The lookup table for the fine motion */

/*
 * Initialize the lookup table.
 */
void getdata()
 {
  int i,j;
```

```
  for(i=0;i<=7;i++)
   resol_p[i]=(int *) malloc(6*sizeof(int));
  for(i=0;i<=7;++i)
    for(j=0;j<=5;++j)
       *(resol_p[i]+j)=0;
  *(resol_p[0]+1)=3;*(resol_p[0]+2)=-1;
  *resol_p[1]=1;
  *(resol_p[2]+1)=-2;*(resol_p[2]+2)=-10;
  *(resol_p[6]+1)=2;*(resol_p[6]+2)=10;
  *(resol_p[3]+3)=*(resol_p[4]+4)=1;
  *(resol_p[7]+3)=-1;
  }

/*
 * This function takes care of the lateral correction.
 */
void  correct(int *offs)
 {
   int  *p_step,step[6],i,j;
   p_step=step;
   for(i=0;i<=5;i++) step[i]=0;
   for(i=0;i<=5;i++)
     for(j=0;j<=5;++j)
      step[i]=step[i]+(*(offs+j)*(*(resol_p[j]+i)));
   move(p_step);
 }

/*
 * This function performs two actions. It moves the end-effector
along
 * the Z-axis and rotate it about the x-axis. param is the variable
 * that specifies which action to perform.
 */
int  resolv(int param,int Pz)
  {
    switch(param)
      {
        case 'Z': move(resol_p[2]);
                  break;
        case 'z': move(resol_p[6]);
                  break;
        case 'R': move(resol_p[4]);
                  break;
        case 'r': move(resol_p[7]);
                  break;
        default :printf("error in function resolv \n");
      }
    position();
    if(*(point1+2) <= Pz)
```

```
    return(-1);
    else
    return(1);
  }

/*
 * This is the main bodie of the program. It insert the peg in
the hole,
 * report the time taken by the operation and then open the griper
 * and moves the end-effector up.
 */
main()
 {
   int j,i,m,n,pos[5],flag,pd,*corr,cons[6];
   double th,thr,arr1[6];
   flag=m=0;
   pd=100;
   thr=8.0;
   th=2.0;
   corr=cons;
   open_com_link();
   open_com1();
   change_frame(3,70);
   display_frame();
   getdata();
   for(n=0;n<=5;++n) cons[n]=0;n=0;
   biostime(1,0);
   while(flag == 0)
     {
       while((arr1[6]=Fz()) >= -thr && flag==0)
         if(resolv('z',pd)<=0)
           flag=10;
         i=j=0;
       while(i == 0)
         {
           if((arr1[0]=Fx()) >= th)
             cons[0]=1;
           if(arr1[0] <= -th)
             cons[0]=-1;
           if((arr1[1]=Fy()) >= th)
             cons[1]=1;
           if(arr1[1] <= -th)
             cons[1]=-1;
           if(cons[1]==0 && cons[0]==0)
           m=0;
           else
            {
              correct(corr);
              m=10;}
           cons[0]=cons[1]=0;
```

```
            if((arr1[4]=My()) <= -2.0)
              {cons[4]=1;
               cons[0]=-1;}
            if(arr1[4] >= 2.0)
              {cons[4]=-1;
               cons[0]=1;}
            if((arr1[3]=Mx()) <= -2.0)
              {cons[3]=-1;
               cons[1]=1;}
            if(arr1[3] >= 2.0)
              {cons[3]=1;
               cons[1]=-1;}
            if(cons[3]==0 && cons[4]==0)
            m=0;
            else
              {
                correct(corr);
                m=10;}
            cons[3]=cons[4]=0;
            ++j;
            if(Fz() >= -4)
            i=10;
            else if(j>=3 || m==0)
              {
                 while(Fz() <= -4 && j<=6)
                   {
                      resolv('R',pd);
                      resolv('r',pd);
                      resolv('Z',pd);
                      ++j;
                   }
                 j=0;i=10;
              }
          m=0;
        }
      }
  printf("\n TIME=%f sec\n",biostime(0,0)/18.2);
  if(spawnl(P_WAIT,"c:\\turboc\\griper.exe","griper.exe","250",NULL)!=0)
  abort();
  *(resol_p[6]+1)=20;*(resol_p[6]+2)=100;
  *(resol_p[2]+1)=-60;*(resol_p[2]+2)=-300;
  while(resolv('Z',130)<=0)
    ;
  for(i=0;i<=7;i++) free(resol_p[i]);
  exit(0);
}
```