HIERARCHICAL TASK DECOMPOSITION AND EXECUTION FOR ROBOT MANIPULATION TASK USING A WRIST FORCE SENSOR

By

Shmuel Kotzev

B.Sc. TECHION, Haifa, Israel

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

MECHANICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

September 1990

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Applied Science

The University of British Columbia
Vancouver, Canada

Date sep 24 1990

DE-6 (2/88)

Abstract

The research developed force-motion strategies and subsequent force and position control algorithms, using a PUMA 560 robot arm and its original controller. A task decomposition methodology has been developed that enables a mechanical assembly task to be subdivided into a series of executable subtasks. By applying this methodology to the assembly of a hydraulic gear pump, a library of special purpose, task oriented, subtask programs were created. Most of these programs, though derived for a pump assembly task, are applicable (when used with appropriate parameters) to other assembly tasks.

Most of the algorithms require force/torque sensory information that is supplied by a JR^3 wrist force sensor. The force control algorithms use that data and system compliance in order to produce new position instructions that are transferred to the controller of the arm. The logic of the control law and system behaviour when contacting the environment, were checked, using the dynamics and compliance of a simplified structure of a robotic arm and its wrist sensor.

A demonstration of the pump assembly task, using the arm, force sensor, controller and the derived library algorithms is an integral part of the thesis.

Acknowledgements

I would like to thank my supervisor, Professor Dale B. Cherchas for the support, guidance and encouragement he provided in the development of this work.

I would also like to thank Doug Latornell, for fruitful discussions and support that eased both my introduction and experimental work in robotics. Also, I would like to thank Alan Steeves and Gerry Rohling for their good advice, patience and support in the use of the department computer system.

Technical assistance was provided by the mechanical and electrical workshops, and especially by Dave Camp and Don Bysouth. I thank them for their assistance in both parts manufacturing and technical support.

The robot arm and force sensor were purchased with financial support from the Natural Sciences and Engineering Research Council of Canada (NSERC). The VAXStation 3200 computer system was provided through an equipment grant from the British Columbia Advanced Systems Institute (ASI).

Nomenclature

$\mathbf{A}$ - a $(n \times 6)$ sensor coupling matrix.

A_t - bolt tensile stress area.

c - clearance ratio $(\frac{R-r}{R})$.

d - nominal diameter of the thread.

$\mathbf{F}_d$ - desired force (control algorithm).

$\mathbf{F}_e$ - force error signal in the control loop $(\mathbf{F}_e = \mathbf{F}_d - \mathbf{F}_s)$.

$\mathbf{F}_i$ - desired preload.

$\mathbf{F}_{nom}$ - vector of nominal measured signals.

$\mathbf{F}_s$ - force sensed by the sensor.

$\mathbf{F}_x$ - applied forces.

$\mathbf{F}_z$ - insertion force.

$\mathbf{G}_s$ - compliance matrix.

$\mathbf{J}$ - the Jacobian of the manipulator.

$\mathbf{K}_f d$ - vector of proportional gains of the switching controller.

$\mathbf{K}_d$ - programmable damping matrix.

$\mathbf{K}_p$ - programmable stiffness matrix (desired elastic behaviour).

$\mathbf{K}_s$ - stiffness matrix.

$\mathbf{K}_x$ - lateral spring constant.

$\mathbf{K}_\theta$ - angular spring constant.

l - insertion depth.

L - lead of the screw.

$\mathbf{M}$ - moment applied by the support.

$\mathbf{q}_a$ - actual joint displacement.

$\mathbf{q}_d$ - desired joint displacement.

$\mathbf{q}$ - error in joint displacement ($\mathbf{q} = \mathbf{q}_d - \mathbf{q}_a$).

r - radius of the peg.

R - radius of the hole.

$\mathbf{S}$ - signal vector of the force sensor.

$\mathbf{S}_{max}$ - max. measuring signals vector.

S_p - bolt proff strength value.

$\mathbf{T}_i$ - torque required to produce a given preload.

$\mathbf{U}$ - lateral error of the compliance center.

$\mathbf{V}_{ax}$ - axial speed of the screw.

w - width of the chamfer.

$\mathbf{x}_a$ - actual task space displacement vector.

$\mathbf{x}_d$ - desired task space displacement vector.

$\mathbf{x}$ - displacement error ($\mathbf{x} = \mathbf{x}_d - \mathbf{x}_a$).

θ - rotational error of the compliance center.

L_g - distance from compliance center.

δ_x - amount of deflection at the tool's tip.

ϵ_0 - initial lateral error.

λ_t - time interval of the discrete system.

μ - coefficient of friction.

$\Delta\theta$ - rotational angle.

Δx - axial translation.

τ - joint torques.

Table of Contents

Chapter 1

INTRODUCTION

An assembly task is an operation of collecting, mating and aligning (in a specified way) two or more three dimensional objects. Performing a mechanical assembly task is a geometric position problem usually solved by highly sophisticated human operation. In order to accomplish the same assembly tasks by a robotic arm (manipulator), the following parameters should be part of the system:

- both gross and fine motion ability of the arm's tip (end effector).
- high speed, high capacity computer hardware utilizing a high level language. The software library must include sophisticated programs that control the manipulator movements, using mostly, a translation of human experience to program instructions (feedback control).
- variety of sensors including force/torque or tactile sensors, a vision system etc. These devices enable the system to verify task completeness, to close control loops and, in very advanced systems - to assist in task decomposition.

In order to actually execute an assembly task, the theory and algorithms were derived for the robotic system and equipment located in the U.B.C. Department of Mechanical Engineering CAMROL Laboratory. A full description of laboratory facilities and equipment is given in chapter 2. As an example, a *Pump Assembly Task* was performed, using the CAMROL equipment.

The system's control is realized hierarchically in several levels. Every control level solves its specific tasks, using special algorithms and hardware. According to Vukobratovic [41] the control hierarchy can be realized in three levels: Strategical control, tactical control and executive control. In order to properly control those levels while executing an assembly task, it has to be decomposed according to the laboratory equipment, the mechanical knowledge (movements, forces etc.), the execution strategy and the description of the task (Fig. 1.1). Task decomposition can be done automatically by a task planner computer algorithm or manually by a human designer. Chapter 3 deals with the ways to manually decompose a task into smaller programs that can easily be executed and controlled. At the end of chapter 3, there is an example of how a decomposition is implemented on a Pump Assembly Task.

Another major objective of the research was to develop a library of necessary force-motion strategies and subsequent force and position control algorithms. Assuming that a library of such programs is stored in the computer memory, the designer can call any desired program in any order to perform a larger overall assembly task. The force control algorithm is part of the execution strategy of the assembly task. Chapter 4 begins with force control review and ends with the description of the algorithms used in this thesis.

The main task - the assembly of a pump, is decomposed to use subtasks (chapter 5), subroutines (chapter 6) and service programs (chapter 7). These are VAL-II, high level programs, based on force information data, received from the JR^3 force sensor. The programs are task level programming: describing the assembly task as a sequence of positional goals of the object. The control of the assembly tasks is based on fine motion strategies applicable (where possible) to the global assembly tasks.

One of the main subjectives of the thesis was to create the programs, based on the description of Pump Assembly Task. Task assembly strategy is the basis of the decomposition and force motion relations and every program is described according to its

function. It is important to understand that although the programs (subtasks, subroutines and service programs) were derived from the description of pump assembly, they are not oriented to that specific task and can be used in any other assemblies. The structure of every algorithm is based firstly on initialization of parameters that are task oriented. Those parameters are received from a special file (initialization file) written by the operator. Using these initialization parameters, the operator can command the algorithm to execute a subtask according to his specifications.

In order to predict the dynamics of simple force-controlled robot system and to understand it's reactions, a force control simulation was created. A description of the simulation and its results is given in chapter 8.

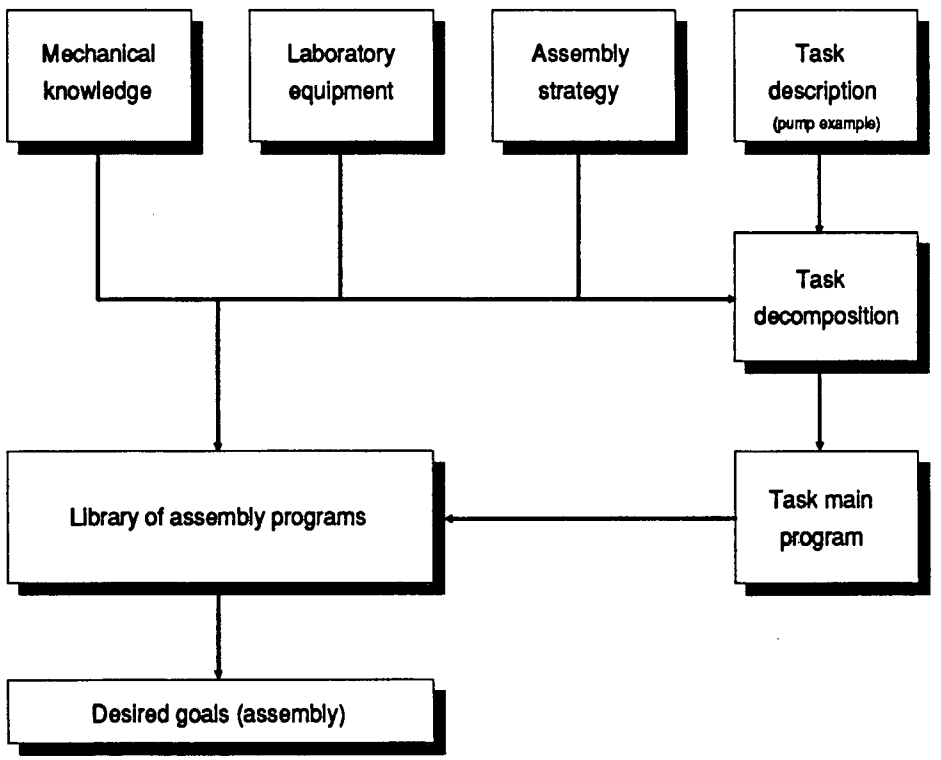

Figure 1.1: Task execution structure.

Chapter 2

SYSTEM DESCRIPTION

The system is built around a PUMA 560 robot arm, its controller, a JR^3 force sensor and a workspace. The chapter gives a basic background about the major devices and parameters of the system (Fig. 2.1), used to accomplished both the experimental and analytical parts of the thesis.

2.1 PUMA 560 ROBOT

Unimation PUMA 560 arms are an advanced computer controlled, widely used class of industrial manipulators. The PUMA robot system is designed to adapt to a wide range of applications. The robot is capable of applying a max. static force of 58 [N] at the tool point and to move with this load in max. tool velocity of 1.0 [m/s] and max. acceleration of 1.0 [g]. The robot arm operating envelope is approx. a radius of 0.9 [m].

Fig. 2.2 shows the basic unit plus the peripherals in use in CAMROL. The robot system includes the following units: the robot arm, controller, software (memory and floppy disk), teach pendant and I/O module. The robot arm executes the instructions transmitted to it by the controller. The user controls the controller manually, by the teach pendant or by running user programs (developed at VAL-II language). The programs can be executed on the local LSI-11/02 microcomputer (located in the controller) or on an external supervisor computer. In all these cases, the controller transmits the instructions from the computer memory to the motors located in the links of the arm. Position data obtained from incremental encoders and potentiometers (located in the robot joints) are

transmitted back to the controller to provide closed-loop control of the arm's motions.

The high-level programming language is VAL-II. Generally, VAL-II software and programs are stored in the computer memory which is located in the controller. Programs may also be stored in a floppy disk or an external computer. This interface may be carried out by matching the Unimation floppy disk drive protocol, using the existing serial communication line of the floppy drive.

Additionally, programs can be executed to implement an interface between the robot and its environment. A peripheral device with a panel switch board, enables the transfer of external input and output signals. Use of these signals in the program enables the programmer to halt a program or branch to another program or subroutine. Communication with sensors or host computers is done through serial ports (RS-232C interface), DMA port or the panel switch board.

2.1.1 Arm Structure:

The arm is a revolute, 6 degrees of freedom (d.o.f) type manipulator (Fig. 2.3) consisting of a trunk (link 0), a shoulder (link 1), an upper arm (link 2), a forearm (link 3), a wrist (link 4) and a gripper (links 5 and 6). Each one of the 6 joints is driven by a geared, permanent-magnet D.C. servo motor, activated by a power amplifier board, mounted in the controller box. The motor systems contains the following devices:

- Integral brakes: a safety feature which is installed in the major axes (joint 1, 2 and 3). When power is removed from the motors, the electromagnetic brakes are activated and lock the joint at the current position. This prevents the arm from collapsing or movements due to accidental (or deliberate) power removal.

- A geared potentiometer is a part of the motor assembly. It supplies low resolution, absolute joint position data. This information, combined with the index signal of

an encoder, is used to initialize joint position during calibration process of every power-on of the arm.

- An incremental optical encoder is connected to the motor shaft. It supplies high resolution rotation data, as well as the direction and absolute angular position of the motor shaft (index signal every 360 degrees). Approximately every 0.875 [msec], the encoder's outputs are compared with the calculated position, and any necessary corrections are generated allowing overall rated repeatability of ± 0.1 [mm].

The PUMA at CAMROL is equipped with an optional pneumatic (710 KPa), computer controlled gripper. The two-way solenoid valve enables full pneumatic force in both the open and the closed positions. Special purpose jaws can be mounted onto the gripper according to the desired task.

2.1.2 Controller and Interfaces:

The controller is the master component of the electrical system. All signals to and from the robot pass through the controller and are used by it to perform real time calculations to control arm movement and position. Ref. [13] gives an essential summary, while Ref. [38] gives a full description of the subject. The controller box is used for:

- Location, protection and connection of the electronics boards and power devices.
- Operation and control panel equipped with operating switches and indicators. This enables the user to switch on/off the unit, the arm, the breaks and the execution of the program.
- Connector base which connects the robot arm, terminal, floppy disk drive and the I/O modules.

Programming is often simplified by constructing the program hierarchically. For manipulator programming, the lowest level usually provides the interface with the manipulator sensors and actuators. In the PUMA, the control scheme of the robot arm is a proportional plus integral plus derivative (PID) controller. It is realized, in the controller box, by two computers hierarchically arranged:

A). The LSI-11/02 supervisory microcomputer performs two major functions. The *user interface* enables on line, bidirectional interaction so interpreting, debugging, scheduling and executing of movement commands and coordinates transformations can be easily done. The second function is the *joint controller interface* that coordinates the controllers of the six joints.

B). Six 6503 microprocessors, joint controllers, are at the lower level of the hierarchy. Each joint is controlled and activated separately by its two servo loops and joint motor. The feedback loops control the position and the velocity variables. Both servo loop gains are constant, a property that limits the flexible dynamic response of the system to varying speeds and payloads.

The system's control is realized hierarchically in several levels. Every control level solves its specific tasks, using special algorithms and hardware. Fig. 2.4 shows the current structure of the robotic control system at CAMROL. According to Vukobratovic (Ref. [41]) the control hierarchy can be realized in three levels (explained in section 3.2):

- Strategical control level.
- Tactical control level.
- Executive control level.

2.1.3 VAL-II Language:

A computer-based system provides an easy definition of a task a robot is to perform. Using a computer-based system for programming and controlling industrial robots provides the following:

a). ability to respond to sensory information.

b). improved performance in terms of trajectory generation.

c). allows operation in unpredictable situations or moving frames of reference.

Several languages have been developed in order to manipulate robot arms in complex tasks. Some of the major languages are: AL, written in Stanford University, PasRo - Pascal and C for Robots, AML and AUTOPASS by IBM, HELP by General Electric, MCL by McDonnell Douglas, JARS by Jet Propulsion Laboratory and VAL-II - a product of Unimation.

VAL-II is a computer-based control system and language designed specifically for use with Unimation Inc. industrial robots (PUMA series). It is a high level, BASIC based, interpreter programming language. VAL-II enables driving the arm by frames manipulation or by motion and speed commands directed to the joints. The executed program can activate other programs through call function and run process control (pc) parallel program at the same time. As a real time system, continuous trajectory computation by VAL-II permits complex motions to be executed quickly, with efficient use of system memory and reduction in overall system complexity. The VAL-II system continuously generates robot control commands and can simultaneously interact with a human operator, permitting on-line program generation and modification.

VAL-II contains the capability of communicating with external systems, using a rigorous communication protocol to ensure the integrity of information transferred between the VAL-II system and the external systems. The communication is possible at three

levels:

- supervisory communication provides a facility for controlling VAL-II system remotely. Any activity which can be performed at the system terminal can alternatively be performed by a remote supervisory system. A standard RS-232C serial line is used for the interface, with a DDCMP protocol.

- program level communication sending output and receiving input from the system terminal, floppy disk and other external devices which communicate serial data.

- real time trajectory modification can be done using data supplied by an external device such as a sensor. This data can be processed 36 times per second while VAL-II continuously informs the external system where the robot tool is located and the status of communication.

In addition to the program level serial communication, VAL-II provides up to sixteen single ended analog input channels and two channels of analog output with hardware option. The following advantages may be gained using those computer interfaces:

- an external system can completely supervise the operation of the VAL-II system (even during program execution).

- sensory generated data may modify the path of the robot while moving (a real-time path control).

- ability to control directly the robot or to create programs in off line and down-line loads (by matching the terminal protocol in the external computer).

- more external data storage capacity (by matching the mini-floppy disk drive protocol in the external computer).

In CAMROL, we used some of these features of VAL-II. The external computer is the VAX Station 3200 which serves as a terminal and a supervisor controller to the PUMA.

2.2 JR^3 FORCE / TORQUE SENSOR

2.2.1 Introduction:

Force/torque sensors, used with robots, are devices used to measure the forces and moments developed between the robot gripper and the environment due to contact. Typical applications are assembly (Ref. [39] and contour tracking (Ref. [12]). Industrial applications involving the use of force sensors for feedback force control are quite rare. This is mostly because force sensor data influences the dynamics of the joint servo loops. Most available controllers (including the PUMA 560 controller) do not allow such low level interaction with the robot. Another reason can be found in the force sensor itself that is a fragile instrument and thus very prone to overloading. Mechanical overload protection is a must, but it is very difficult to achieve it properly.

The following are the optional locations to place force sensors (Ref. [39]):

- in the joint actuators - measures torque/force of the actuator and used mainly by master/slave manipulators.
- in the interface between the last joint and the gripper. A multi component force wrist sensor.
- on the gripper fingers, mainly by means of strain gauges applied to the fingers.
- in the contact area - tactile sensors (artificial skin sensors) placed between the fingers and the grasped object.
- in the robot environment - in the table or the mounting fixture.

The Puma robot has a wrist force sensor plus electronic system from JR^3 company.

A wrist force sensor is an elastic (metallic) structure that deforms under applied forces. The structure is usually mapped with strain gauges that have proven to be the smallest, simplest, cheapest and reliable transducers for use in robot force sensors. The produced output voltage is proportional to the strain in the structure (assuming operation within its elastic range). The raw analog signals of the strain gauge bridges are transferred to the electronics system where they are amplified and processed. Additional processing of the output data is made in the robot controller including calibration, decoupling and coordinate transformations.

The correlation between the input force vector (to the sensor) $\mathbf{F}_s$ (3 force and 3 torque variables) and the inner signal vector of the sensor $\mathbf{S}$, is the output of n different strains measurements output and is given by:

$$\mathbf{S} = \mathbf{A} \cdot \mathbf{F}_s \tag{2.1}$$

where $\mathbf{A}$ is a $(n \times 6)$ sensor coupling matrix. For $n = 6$ the force vector can be resolved by the decoupling matrix $\mathbf{A}^{-1}$:

$$\mathbf{F}_s = \mathbf{A}^{-1} \cdot \mathbf{S} \tag{2.2}$$

If the sensor is *mechanically decoupled* ($\mathbf{A}^{-1}$ is diagonal) then every measuring component is proportional to only one force component. Thus the quality of the force sensor is reflected by the form of its decoupling matrix (a perfect diagonal $\mathbf{A}$ matrix can't be obtained in practice). The presence of nonzero diagonal elements cause the cross sensitivity of the sensor. Other important features of the sensor are:

a) the measuring signals are to reach their max. value $\mathbf{S}_{max}$ when all force components reach their nominal values simultaneously:

$$\mathbf{S}_{max} = \sum_{j=1}^{6} a_{ij} \mathbf{F}_{nom} \tag{2.3}$$

b) to receive the same measuring accuracy on all components, each force component should contribute equally to the measuring signal i.e.

$$a_{i1}F_{1n} = a_{i2}F_{2n} = ... = a_{i6}F_{6n} = \frac{1}{6}S_{in} \tag{2.4}$$

2.2.2 JR^3 Force Sensor:

The force-moment sensor used with the Puma robot is a standard JR^3 Universal Force-Moment Sensor System consisting of:

- JR^3 monolithic six-degree-of-freedom force sensor.
- JR^3 Intelligent Support System That contains:
    1. signal conditioning board.
    2. data acquisition board.
    3. processor board.

Due to forces/torques acting on the sensor the strain gauges change their response, producing small variations (milivolts) in the measured voltage of the system. The raw analog signals are transferred to the electronics box, where they are amplified, filtered, digitized (12 bit A/D) and processed (digital filtering, load envelope monitoring, etc.). Onboard shunt resistors combined with calibration software offer automatic drift compensation. Output can be received as discrete I/O, high-speed parallel interface or through two serial ports.

2.3 COMPLIANCE DEVICES

2.3.1 Remote Center of Compliance (RCC):

The RCC is a mechanical multi axis passive compliant device. While executing assembly tasks in the CAMROL laboratory, the RCC is mounted between the force sensor (or the linear compliant device) and the gripper as can be seen in Fig. 2.5. The RCC was designed to ease assembly of closefitting objects in spite of lateral or angular misalignment by correcting position and angular errors. Typical applications are rivets assembly, screwing bolts, bearing installation etc.. The RCC is widely used in industry due to its price, simplicity of installation and its repeatability and reliable response.

The construction of the RCC is a combination of two mechanical parts: translational and rotational. The translational part allows lateral motion while the rotational part allows angular movement of the end-effector. The flexibility of the LORD RCC Series devices is provided by laminated elements made up of elastomer and steel shims. In compression these elements are much stiffer than in shear. This design enables the laminated element to respond correctly to lateral forces or moments.

The preferable working point for accurate assembly tasks while using RCC, is at its center of compliance or elastic center. It is a specific projected point along the tool axis (its exact location is a function of the structure of the RCC). For part mating, it is ideal if the stiffness matrix (of the part and the structure supporting it) becomes diagonal at the tip of the part. At that point, force/deflection behaviour of the structure becomes decoupled i.e. pure rotation will occur due to applied moments and pure translation will occur due to lateral contact forces.

The major advantage of the RCC is that it absorbs both lateral and angular errors plus twist errors about the insertion axis. Its disadvantages are its limited operating range (suitable only for end-effector/part combination of a single length) and its limited

operating speeds. The IRCC is a combination of a force/torque multi degree of freedom sensor, built into the mechanical compliant structure of the RCC. The RCC in CAMROL is from the LORD company.

2.3.2 Linear Compliance Device:

For a stiff manipulator and stiff environment, part mating introduces high level forces when a position error occurs (the forces are function of the size of the error and the stiffness of the mating parts). Position errors can be introduced by both the tool and the mating object. Tool position error is introduced due to matrix manipulation residuals (locations and inverse kinematics), encoders resolution, tool/gripper misalignment (changing in the relative position due to external forces) etc.. The location of the mating surface is imprecise because of bad positioning in the part location area and unprecise manufacturing.

Special perpose precise parts are partial solution but are impractical. Practical ways to overcome the problem are fine arm motion and/or low compliance.

a) The ability to use fine motion is part of the robot performance and it is activated using process control program with small gains (a pc program runs parallel to the main program and can altered the arm's trajectory - in this case, according to force data). These gains are used whenever the tool is in the neighborhood of another part.

b) The contact forces are function of the dynamic parameters and of the compliance of the arm, sensor and the environment. Decreasing the stiffness of the end effector (or the assembly base) enables the use of higher dynamic parameters (velocity, acceleration) of the arm, without increasing the contact forces. The RCC gives compliance to the system in the $(X-Y)$ plane, while an additional compliant linear device used in the Z direction completes the e-e compliance behaviour. This structure reduces the band width of the system, but in an assembly task, the important parameters are low contact forces and

flexible operation, and after it comes high speed operation.

The added device (schematically represented in Fig. 2.5) was based on 3 linear bearings and a soft spring. The RCC can comply to torques about X, Y and Z axis and to forces in the $X-Y$ plane. The linear device completes the compliance of the system in the Z direction, but was found not to be necessary for the demonstration (Sec 3.4) for slow execution.

Figure 2.1: System structure and workspace.

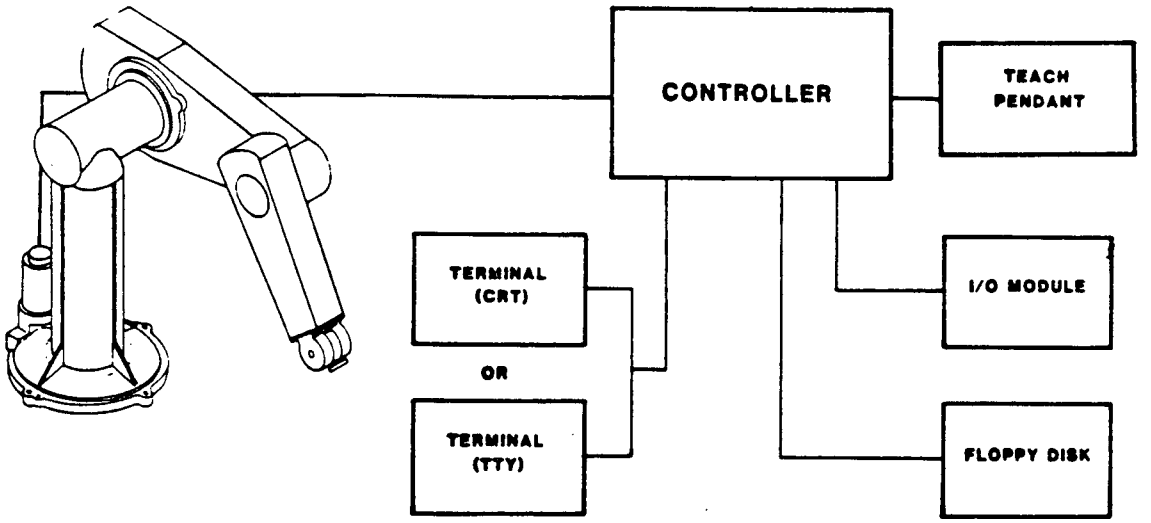

Figure 2.2: PUMA 560 robot system.

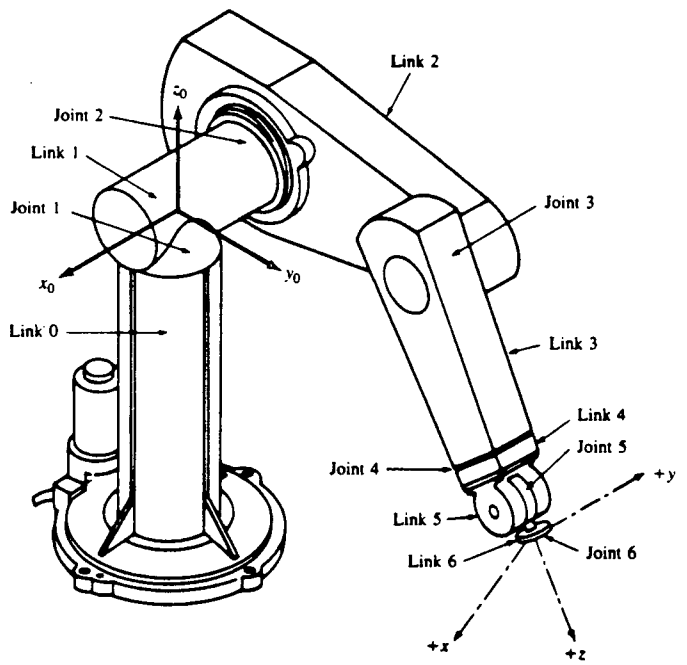

Figure 2.3: PUMA 560 robot arm.

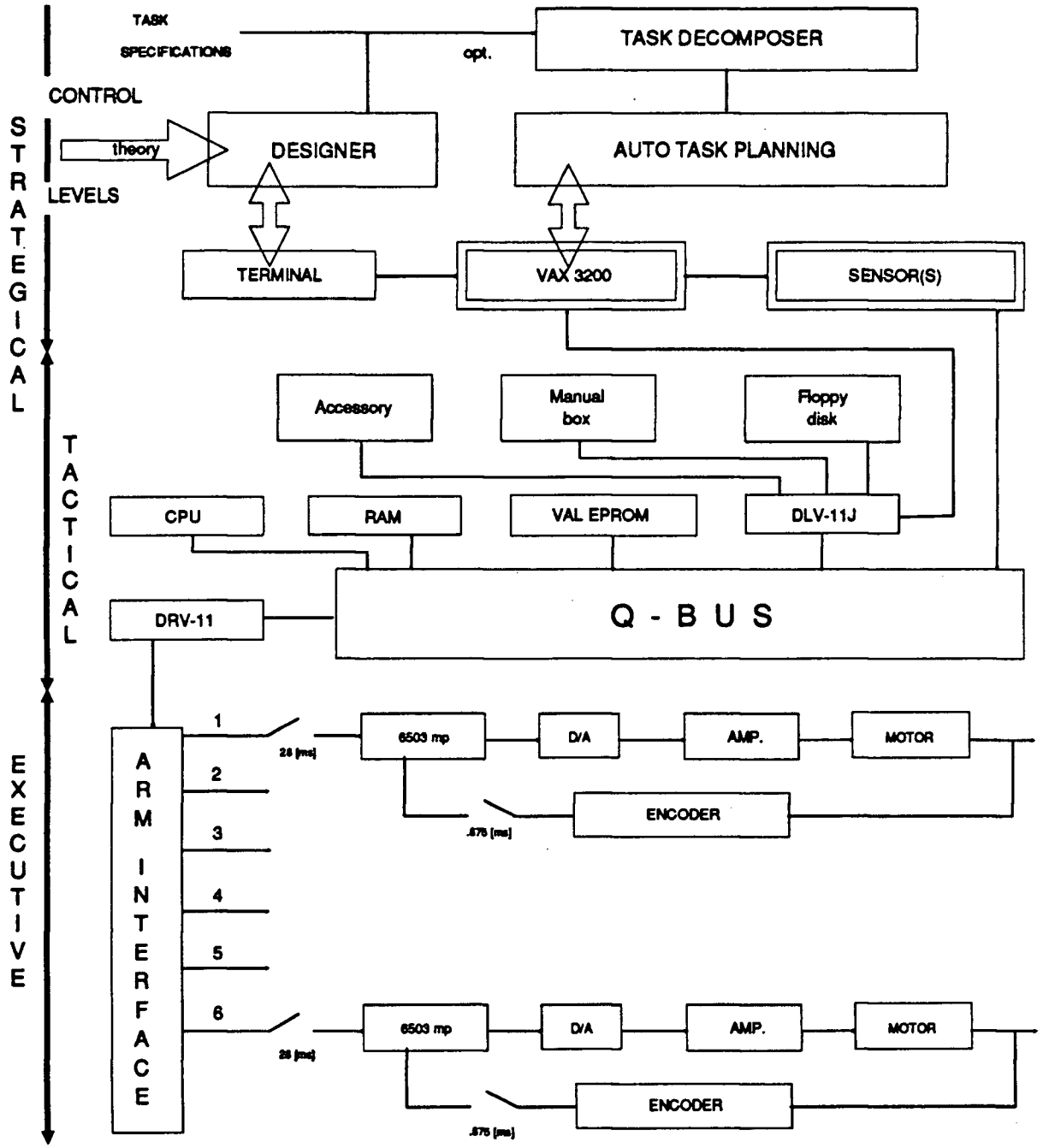

Figure 2.4: Structure and hierarchy of the CAMROL robot system.

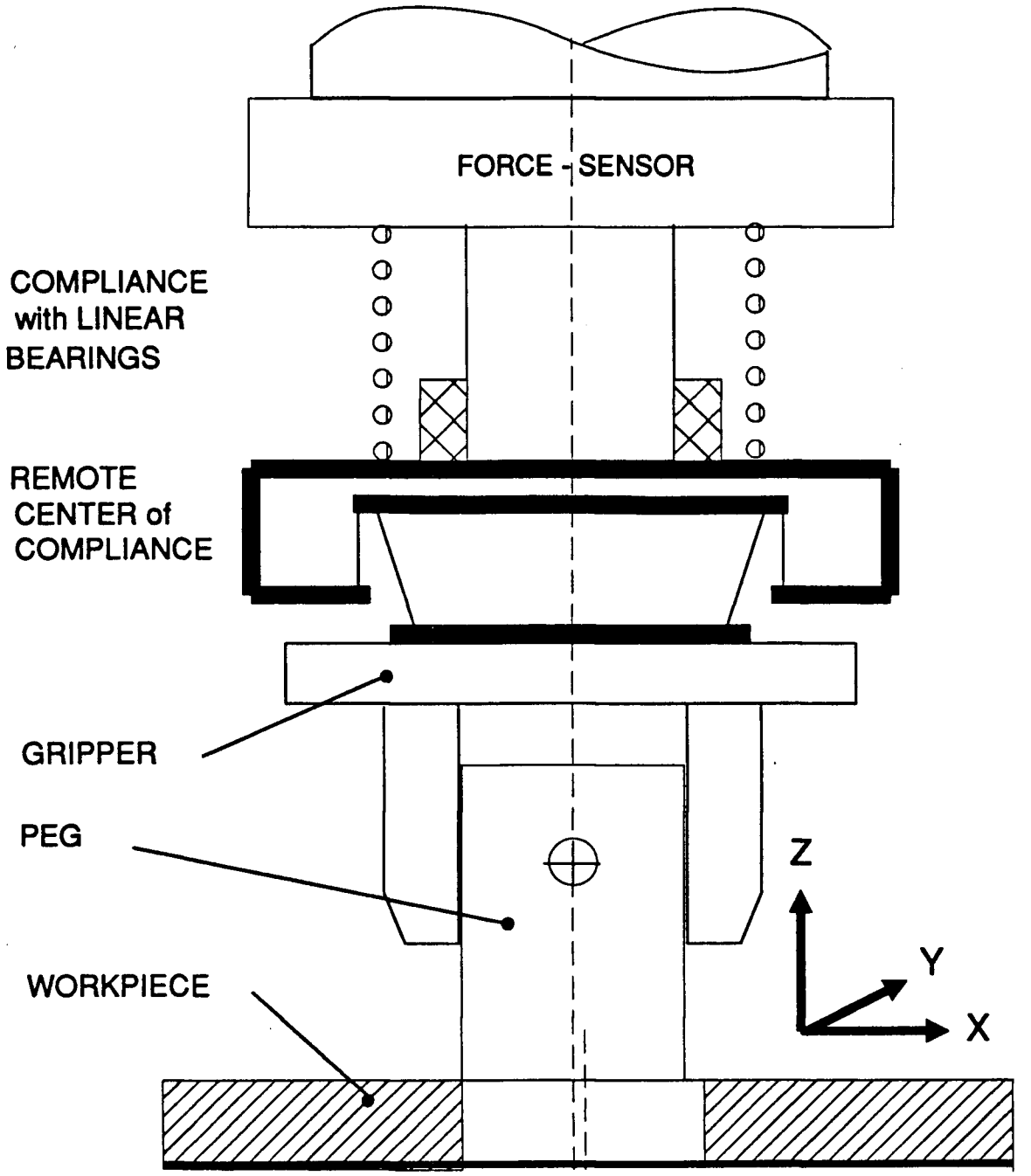

Figure 2.5: Structure of the end effector for assembly task.

Chapter 3

TASK DECOMPOSITION

3.1 INTRODUCTION

One of the objectives of the research was to develop the force-motion strategies and force/position control algorithms for selected assembly subtasks. Those programs are part of a software library that can be used to construct many other assembly tasks, using a manipulator equipped with force sensor. The programs from the library are activated according to the order received by decomposing the main assembly task. This chapter describes the logic behind the decomposition of an assembly task into a set of programs that enables the actual assembly by the arm. The chapter begins with literature review and describes the decomposition hierarchical structure and the way it functions.

A specific task, i.e. Pump assembly, was chosen to be in the background for the library of assembly subtasks. The programs derived in the thesis are those that are needed to execute this assembly task. The next sections describe the way the main task is decomposed using subtasks, subroutines and service programs. Here, task planning and the way it can be executed, is based on CAMROL's facilities, on the mechanical structure of the pump and on assembly logic. It is important to emphasize that the *library structure* developed and described in the following chapters is not exclusively for the Pump Assembly. The subtasks are part of a general library and can be used to generate any special purpose assembly task. The only part that is task oriented is the program **PUMP** and its related **Initialization files**. In the future, for other new

assembly tasks, it is likely, that some new programs (subtasks, subroutines and service programs) may be needed. Building those new programs in the same structure logic that is developed in this thesis, will allow the designer to use the other programs of the library together with the new programs.

The execution of an assembly program involves contacts between the end effector and the environment. The surface in contact introduces kinematic constraints that modify the motion of the manipulator. The use of force sensory information and force/position control, enables the constrained manipulator to regulate force and torque reactions between the end effector and the environment while simultaneously moving the arm to the desired position and orientation. Force control and force/position relations are the heart of the assembly algorithms therefor, a review and development of the subject is presented in the next chapter. Based on both subjects: task decomposition and force control algorithms, the following chapters describe the way pump assembly task is implemented.

3.2 LITERATURE REVIEW

Sophisticated, intelligent tasks can be executed by means of hierarchically intelligent control systems. Manipulators like Telerobots are designed to be controlled both directly by a computer and remotely by a human. This supervisory control, allows the operator to apply his intelligence to the task without having to maintain continuous control. The use of Hierarchically Intelligent Control can be applied (Ref. [35]) based on the following principle: increasing intelligence (additional sensors, suphisticated software) enables the decreasing of the precision (arm, workspace). It can also lower human activity in the manipulator operation. In this chapter, Task Decomposition control level is discussed. It performs real time planning and task monitoring as well as task decomposition of high level goals into low level actions.

According to Vukobratovic (Ref. [41]) the control hierarchy can be realized in several levels, so each control level solves its specific task. The one usually adopted for robotic control system can be realized in the following three levels (Fig. 2.4):

- *Strategical control level* defines mostly trajectories and system actions by using pre-defined (off line) information. In advanced systems, this level uses sensory information to solve problems in real time using high level programing languages.
- *Tactical control level* is mostly implemented by inverse kinematics solutions that use desired position and orientation of the end-effector plus joint and link parameters of the robot in order to calculate the corresponding desired joint angles.
- *Executive control level* has to realize the positions of the robot joints which are imposed by the higher, tactical control level. This level in the PUMA control system has both position and velocity servo loops which critically damp each joint according to the program's specified speeds.

In Ref. [35] (Saridis), a three level hierarchical structure is presented: including the models of the upper two. The three levels are:

1. The upper level is the organization level that performs general knowledge processing tasks with little or no precision. This level performs five sequential functions: machine reasoning, machine planning, decision making, feedback and long-term memory. It formulates complete and compatible plans and decides about the best possible plan to execute the user requested job.
2. The *coordination level* performs specific knowledge processing tasks, by using specific coordinators, each performing its own pre-specified functions. Its purposes are to coordinate the individual tasks, to assign penalty functions and to optimize the performance of the overall plan.

3. The *execution level* is composed of specific execution devices associated with each coordinator.

Ref. [2] (Albus), describes three legged hierarchical architectures that are executed in parallel and have some levels that are function of the desired task. In this case, the decomposition module is not the upper level, instead, each of its levels is executed in parallel to the two other modules: world modeling and sensory processing. In each operational level, the task decomposition module is using world modeling information that is continuously updated by the third module. This is a dynamic way to activate the manipulator in an environment that is changed during operation or in an unpredicted environment.

In each level, the task decomposition module consists of the following functions:

1. *job assignment manager* is responsible for partitioning the task into distinct jobs to be performed by the planner and the executor mechanisms. At some upper levels it may also assign physical resources against task elements.

2. one or more *planners* that are responsible for decomposing the jobs into sequence of planned subtasks. The planner hypothesizes some action or series of actions, and the world model predicts the results of the action(s).

3. one or more *executors* that are responsible for successfully executing the action prepared by the planner.

The disadvantage of the architectural models presented in [41] and [35] is that they do not consider the possibility of unpredictable events during plan(s) execution. While in the structure presented in [2], the last three inner parts of the decomposition module are arranged in that hierarchical structure, but they are embedded in another structure

that is updated using real time sensory information. It can concluded that the overall structure in the last case is more developed to act in less predicted workspaces.

3.3 ASSEMBLY STRATEGY AND ALGORITHM STRUCTURE

An assembly task is introduced to the system by its specifications. In order to execute it successfully using the robotic system, the specification has to be decomposed to the level of robot instructions. Task decomposition can be done by automatic task planning that is based on artificial intelligence or manually by the designer. Operational programs which are derived on the basis of this decomposition, are built from instructions that can be understood by the arm's controller. This thesis deals with manual decomposition which can be implemented for automatic multi task planning by adding the necessary AI modules.

While decomposing an assembly task the designer has to consider the following steps:

- *Task description:* Tasks are generally specified in terms of the desired motions and contact forces of the end-effector. The planner has to create the best strategy for those parameters taking the following into account: assembly goals, environment (workpiece structure and locations of target points and obstacles), end-effector structure and equipment, forces/motions relations and position/velocity states.

- *Control and Stability:* The best ratio between speed and accuracy and the best error handling that can be achieved by applying correct control. The type of sensor, the dynamic (arm, load and environment) and the kinematics of the system must be considered in choosing the control strategy. For example, in dealing with an environmental sensor (touch) a feasible control strategy will be based on constraints.

An assembly task is a process of mating two workpieces. One workpiece is fixed in the workspace while the other is manipulated by the robot end effector. In fact, the

motions of the arm are constrained due to contact with other surfaces. The *natural constraints* are the result of the particular mechanical and geometric characteristics of the task configuration (natural position or force constraints). They arise naturally from a particular contacting situation and have nothing to do with the task planning. While the natural constraints avoid movement of the end-effector, the *artificial constraints* specify the desired motion or force application. The artificial constraints will produce the desired goal trajectory if, and only if, the goal trajectory is the unique solution of the combined artificial plus natural constraints. Returning to assembly strategy, the planner can describe the desired procedure for the task in a sequence of artificial constraints. In general, for many solutions, a set of artificial constraints which are orthogonal to the natural constraints will be chosen. This may be defined thus:

1. a *generalized surface* is defined by position constraints along its normals and force constraints along its tangents. The position and force constraints are the components of the natural constraints. They are sometimes declared as "velocity equals zero" and "force/torque" constraints.

2. a *constraint frame* is a frame in which its origin is located according to the way the planner controls the execution of the task. The frame may be fixed in the environment or may move with the end-effector.

The desired motion in a specific subtask can be defined by specifying a list of constraints as the basis of the control algorithm.

The structure of an assembly task program has to be simple to understand and easy to operate. This, and the limited memory space in the control box of the robot compel an hierarchical structure. Fig. 3.2 schematically represents this structure, derived here for the example task, shown in Fig. 3.1. The following is an explanation of the hierarchy levels (from lowest to highest) of the programs in the library:

- *SERVICE PROGRAMS:* at the lowest level of the pyramid there is a group of special purpose programs. They are used for tasks like communication (controller to force sensor electronic box or controller to I/O box), display of information, force and tool transformation, overload checking, signals activating and resetting etc.

- *SUBROUTINES:* there are limited number of basic operations that are the building blocks of a special purpose assembly task. Using a fixed structure subroutine, with appropriate parameters derived from task description, enables actual movement of the arm in the desired trajectories. Assembly subroutines whose controller is based on force information, extensively use Process Control (pc) programs. In this program structure the main subroutine controls arm movements according to position control while, running in parallel, the pc program controls forces by real time alteration of arm movement.

- *SUBTASKS:* under this title there are repeatable operation oriented programs that enable easy understanding and decomposition of the main task. Each subtask is responsible for a series of movements from one target point to another. All the intermediate points are related to those two points. During the execution of a subtask, the end effector configuration can be changed only once. Every subtask contains an initialization file that assigns desired values to a list of execution subroutines (called by that subtask). The subtask calls the subroutines according to the logic flow of the task, using the structured constructs supplied by VAL-II.

- *MAIN TASK:* every assembly task has one main task that controls its flow from the beginning to its end. This program is broken into initialization programs and subtasks based on task specifications. Into the main task the designer can add check points, decision points and alternative subtasks.

The decomposition of any assembly task should follow the same hierarchy as described above.

Task decomposition is the way mechanical knowledge, laboratory equipment and assembly strategy are applied to a task description in order to achieve the desired goals. The output of the decompositions are two task oriented programs: the initialization file and the main program that activates the library programs (described in the following chapters). In the library, VAL-II instructions are grouped together to form programs that follow the logic of the assembly strategy. The programs are constrained by VAL-II instructions and limitations, by the special structure of the library and by the memory limitations of the controller. During execution, the memory contains all the declared parameters to be transferred from one program to another. The last values will be used unless they are updated. This can cause unexpected problems (eq. null parameters for locations). A solution (a good programing habit) is to give values to each parameter at the beginning of each program. Fig. (3.3) shows the way it is done in the thesis: each subroutine has a parametric file that contains basic values for all the parameters used in that program. Each *initialization file* calls other *Parametric files* (according to the required subtask). This loads a set of parameters with certain values. If for a specific task some values have to be changed, those changes are introducing to the system *after* calling the parametric file and this last declaration remains during program execution. In this way all the parameters are listed and declared before running the programs and the initialization files are relatively small (contains only the changes)

For example, a better understanding of this programing structure can be achieved by carrying out the simple example illustrated in figure 3.1. In this assembly task, the robot manipulator has to screw four bolts into the block located on a table. Task description includes locations of the block's threads, bolt pick up point, bolt dimensions and weight, maximum tightening torque for each bolt and gripper's dimensions and weight.

The basic gross operations that an assembly operator (human or robot) has to do is to **Pick** up a bolt from the feeder and to **Screw** it into one of the threads located on the block. Then, to **Pick** another bolt, change its target point and **Screw** it into another thread and so on. These two operations (subtasks) are repeated 4 times. The **Pick** subtask is always the same while in the **Screw** subtask the target point and the location of the threads change. Fig. 3.2 shows the block diagram for this operation. The main task is decomposed and executed by a series of initialization files and subtasks. In the first initialization file - **Pin1.ini1** all the parameters for the first **Pick** subtask are included, the same with **Pin1.ini2** file that holds the parameters for **Screw** subtask. If the pick up locations of all the bolts in the feeder are the same, no new initialization is needed before subroutine **Pick** is executed. But, as on the block, there are four different thread locations, an initialization file has to be included before execution of the **Screw** subroutines. In this structure, checking or decision points are not included in the main task, they are presented in the lower subroutine level.

Entering the subtask level in the algorithm structure (Fig. 3.2), every subtasks is built from subroutines that describe a lower operational level. The following is the operational description of subroutine sequence within Pick and Screw subtasks:

1. **Subtask Pick** - from any location of the end effector

   (a) **Move** the arm to a point above the screw (both bolt location and the height above it were received from initialization file).

   (b) **Approach** rapidly in gross movement towards the bolt and near the contact point, change the approaching motion to fine motion, force control approach.

   (c) **Grasp** the bolt and weight it to verify that the grasp operation were successfully completed. If the grasping failed the program will run **Pick** subtask again.

(d) **clear** the area to a point above the feeder.

2. **Screw** - from any location of the end-effector

   (a) **Move** the arm (and the screw) to a point above the desired thread location on the block (both thread location, the height above it and bolt size were received from initialization file).

   (b) **Approach** subroutine is executed to receive low impact contact between the tip of the bolt and the block surface.

   (c) **Search** program is executed to verify that the bolt is located within the thread borders (indicated by high normal forces from the thread's edges). If not, **Search** will move the gripper in a search pattern (while increasing size of the square ribs) until the thread will be located.

   (d) **Rotate** program will rotate the screw until the desired torque will be reached while complying gripper position and orientation to forces/torques reactions.

The subroutine level makes use of the service programs. The **Approach** subroutine, for example, approaches the surface using location transformation of the tool and gripper sizes and is terminated when desired normal forces are reached. The information needed is available using service programs like **JR3.DAT** to receive force information or **Tip** for location transformations.

It can be seen that in the same main task - subtasks, subroutines and service programs are executed again and again. It can be done because of the use in the initialization files and the ability to change parameters during program flow. The structure logic, the programs and the way their parameters are derived are a major element in this research and will be discussed in the following chapters.

3.4 PUMP ASSEMBLY DEMONSTRATION

Pump assembly was chosen as applicable to represent and demonstrate robot performance, using force sensory information. This task is a common industrial task with all the basic subtasks that can be found in many other mechanical structures. The *external hydraulic gear pump* type is a common pump, and the one used as an example in this thesis, is shown schematically in Fig. 3.5. During demonstration, the following parts are manipulated by the arm: Gear1 (2), Gear2 (3), Top (4) and 4 Bolts (5). The subtasks in the thesis, are derived in order to assembly those parts. In the figure, like in the workspace, the pump is mounted on a base, with two pins to prevent its movement.

The working space that lies inside the operating envelope (ref. [38]) of the arm is divided into an *assembly area* (workspace) and *parts location* (feeder). The base of the pump is mounted on an aluminum base in front of the robot. It simulates the part that is moved to its position by a mechanical feeder or a belt conveyer. The parts to be assembled are fed to a nearby location by another feeder. After the assembly of the last part by the arm, the conveyer is moved with the assembled pump and new empty pump base is moved in front the robot to repeat the process. In the laboratory both the base of the pump and the parts are installed on aluminum bases that are located on a wooden (compliant) table. All the parts are located with relatively fixed positions, so no Search pattern is used to locate their position and orientation.

The arm is equipped as described in chapter 2. The opening range of the gripper was adjusted so it can move all the parts. The devices mounted on the e-e were measured and their relative position from the base of the force sensor is used as an input for position and force transformations (see chapter 6).

The following table summarizes arm movements and check points and shows the basic logic behind the main assembly decomposition. Every operation is described and

its relating subtask program and initialization file are enclosed. Each subtask begins or ends with an arm movement (with or without a tool or a part). The pattern of movement includes a horizontal movement that is always made in a safe place (free space) located above the assembly area. **Approach** and **Clear** (depart) horizontal movements connect between the free space and the workspace (or the parts location).

Pump Ass. Subtasks Description:	*Subtasks:*		
1. GEAR1:			**S**
1.1 Move to gear1, grasp it and check	1.1.1 GEAR1.INI1	**S**	**E**
the grasping.	1.1.2 PICK	**U**	**R**
1.2 Insert gear1 into its place in the	1.2.1 GEAR1.INI2	**B**	**V**
base.	1.2.2 INSERT	**R**	**I**
2. GEAR2:		**O**	**C**
2.1 Move to gear2, grasp it and check	2.1.1 GEAR2.INI2	**U**	**E**
the grasping.	2.1.2 PICK	**T**	
2.2 Insert gear2 into its place in the	2.2.1 GEAR2.INI2	**I**	**P**
base.	2.2.2 INSERT	**N**	**R**
3. PUMP TOP (COVER):		**E**	**O**
3.1 Move to Cover, grasp it and check	3.1.1 TOP.INI1	**S**	**G**
the grasping.	3.1.2 PICK		**.**
3.2 Place the cover on the gears (two pins	3.2.1 TOP.INI2		
insertion) and twist to verify location.	3.2.2 INSERT		

PUMP Ass. Subtasks Description:	*Subtasks:*		
4. PLACING SCREW1:			
4.1 Move to Screw1, grasp it and check	4.1.1 SCREW1.INI1		
the grasping.	4.1.2 PICK		**S**
4.2 Move to the desired hole, place it	4.2.1 SCREW1.INI2		**E**
and make 3 turns.	4.2.2 PLACE		**R**
5. PLACING SCREW 2:			**V**
5.1 Move to Screw2, grasp it and check	5.1.1 SCREW2.INI1	**S**	**I**
the grasping.	5.1.2 PICK	**U**	**C**
5.2 Move to the desired hole, place it	5.2.1 SCREW2.INI2	**B**	**E**
and make 3 turns.	5.2.2 PLACE	**R**	
6. PLACING SCREW 3:		**O**	
6.1 Move to Screw3, grasp it and check	6.1.1 SCREW3.INI1	**U**	**P**
the grasping.	6.1.2 PICK	**T**	**R**
6.2 Move to the desired hole, place it	6.2.1 SCREW3.INI2	**I**	**O**
and make 3 turns.	6.2.2 PLACE	**N**	**G**
7. PLACING SCREW 4:		**E**	**R**
7.1 Move to Screw4, grasp it and check	7.1.1 SCREW4.INI1	**S**	**A**
the grasping.	7.1.2 PICK		**M**
7.2 Move to the desired hole, place it	7.2.1 SCREW4.INI2		**E**
and make 3 turns.	7.2.2 PLACE		**S**
8. PICK TOOL:			
8.1 Move to Tool1, grasp it and check	8.1.1 TOOL.INI1		
the grasping.	8.1.2 PICK		

PUMP Ass. Subtasks Description:	*Subtasks:*		
9. SCREW 1:			
9.1 Move to Screw1, grasp it and turn	9.1.1 SCREW1.INI3		
4 turns (controlled torque).	9.1.2 ROTATE		
10. SCREW 2:			**S**
12.1 Move to Screw2, grasp it and turn	10.1.1 SCREW2.INI3		**E**
4 turns (controlled torque).	10.1.2 ROTATE		**R**
11. SCREW 3:		**S**	**V**
11.1 Move to Screw3, grasp it and turn	11.1.1 SRCEW3.INI3	**U**	**I**
4 turns (controlled torque).	11.1.2 ROTATE	**B**	**C**
12. SCREW 4:		**R**	**E**
12.1 Move to Screw4, grasp it and turn	12.1.1 SCREW4.INI3	**O**	
4 turns (controlled torque).	12.1.2 ROTATE	**U**	
13. APPLY TORQUE:		**T**	**P**
13.1 Move to Screw1, Screw2, Screw3 and	13.1.1 REPEAT.INI1	**I**	**R**
Screw4 and tighten them to the	13.1.2 ROTATE	**N**	**O**
desired torque as indicated in		**E**	**G**
REPEAT.INI1.		**S**	**R**
14. REMOVE ASSEMBLY:			**A**
14.1 Move to the assembly, grasp and	14.1.1 ASSEMB.INI1		**M**
check the grasping.	14.1.2 PICK		**S**
14.2 Move to desired place and place	14.2.1 ASSEMB.INI2		
the assembled pump.	14.2.2 PLACE		

Fig. 3.4 shows the beginning of the structure that is used in order to execute the **Pump Assembly Task**. The main program calls the initialization files and the subroutines **Pick** and **Place** for Base positioning and **Pick** and **Insert** for Gear1 insertion. If **Insert** subtask is taken as example from the table, it is used to insert both Gear1 and Gear2. The insertion of Gear1 is relatively simple: the shaft of the gear is inserted to its bearing in a "peg in a hole" routine, while considering possible interaction between the pump's walls and the tooth of the gear. Inserting the second gear is a little different because while the shaft is inserted and the walls interfere, the teeth have to match during the linear movement. A relative rotation is added to the insertion program to overcome the problem. Those and other details are not included in the table - chapter 5 covers the subtasks in details.

Every subtask contains a list of subroutines, part of them are being executed by triggering a software switch in the initialization program. The following table summarizes all the programs contained in the assembly library which allow successful execution of **Pump Assembly Task**:

Main Task	*Subtasks*	*Subroutines*	*Service Programs*
Pump-	Pick	Approach	PCstop
Assembly	Place	Grasp	Force information
	Insert	Search	Process control programs
	Screw	Comply	Initialization and
	Check	Clear	Parametric files
		Place	Overload checking
		Move	Weight

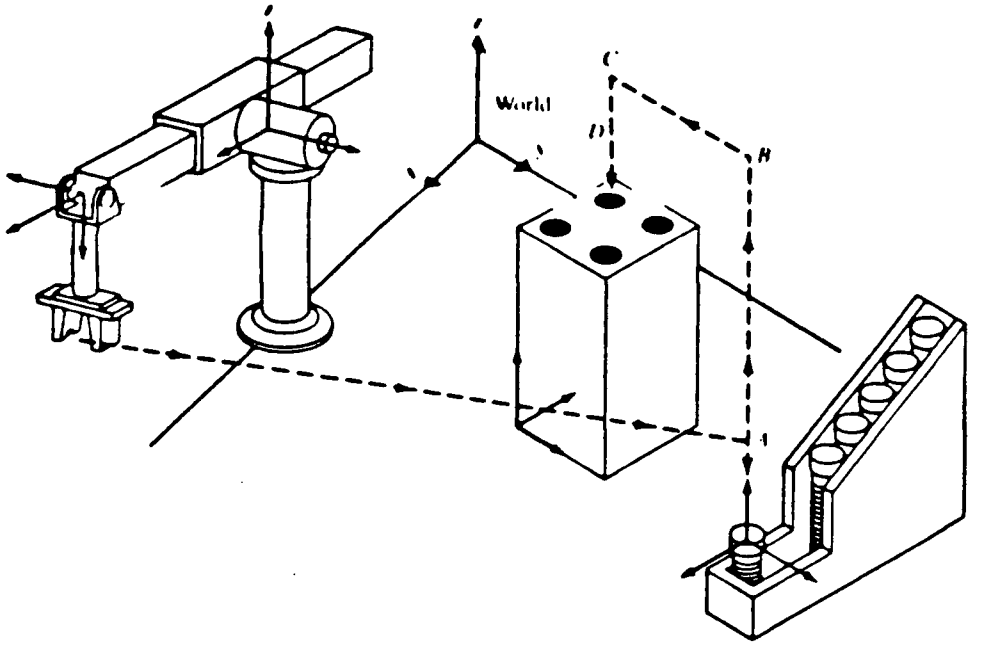

Figure 3.1: Execution of Block assembly by a robot.

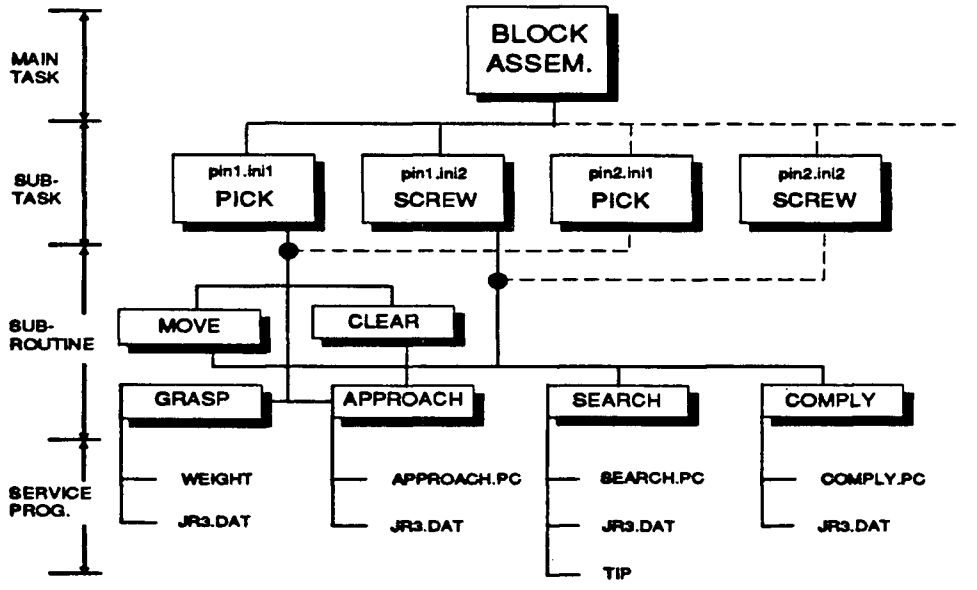

Figure 3.2: Algorithm structure using Block Assembly example.

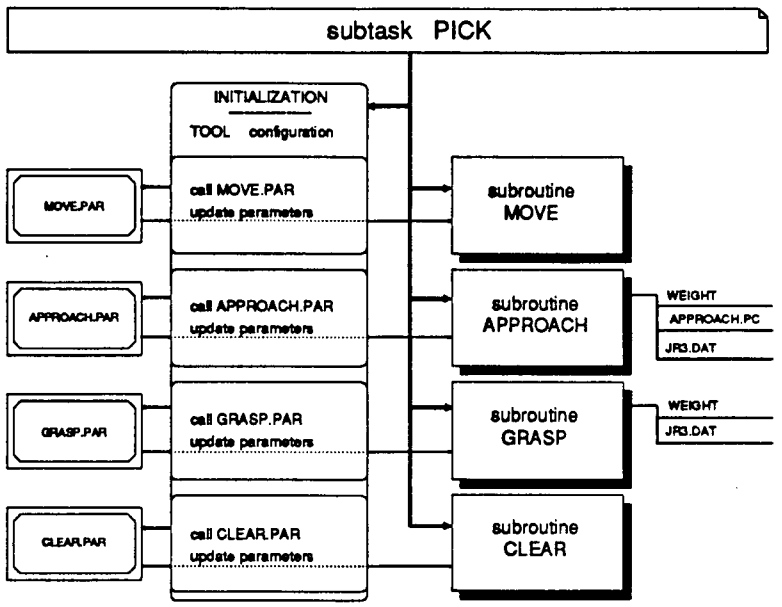

Figure 3.3: Decomposition of subtask Pick.

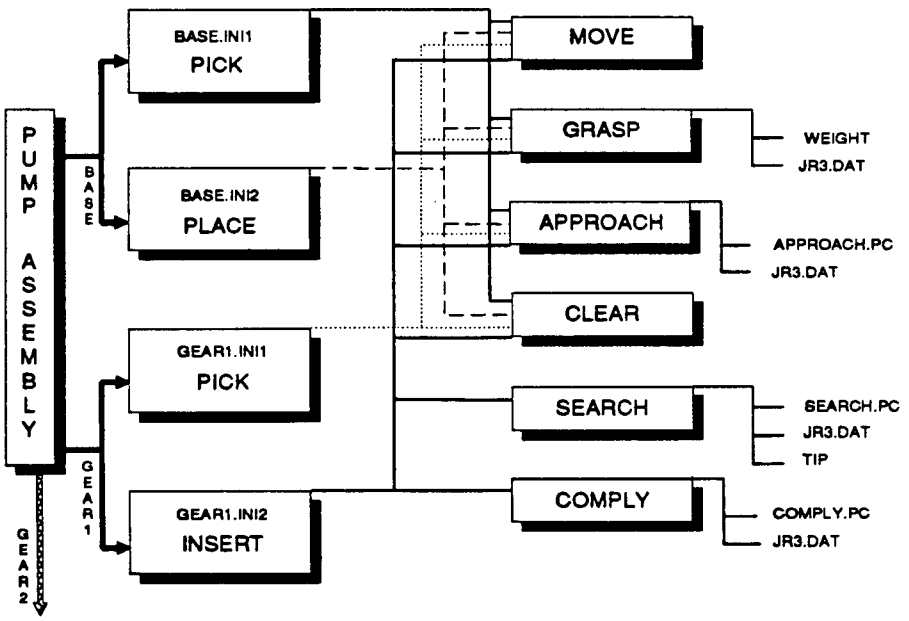

Figure 3.4: Beginning of Pump Assembly Task.

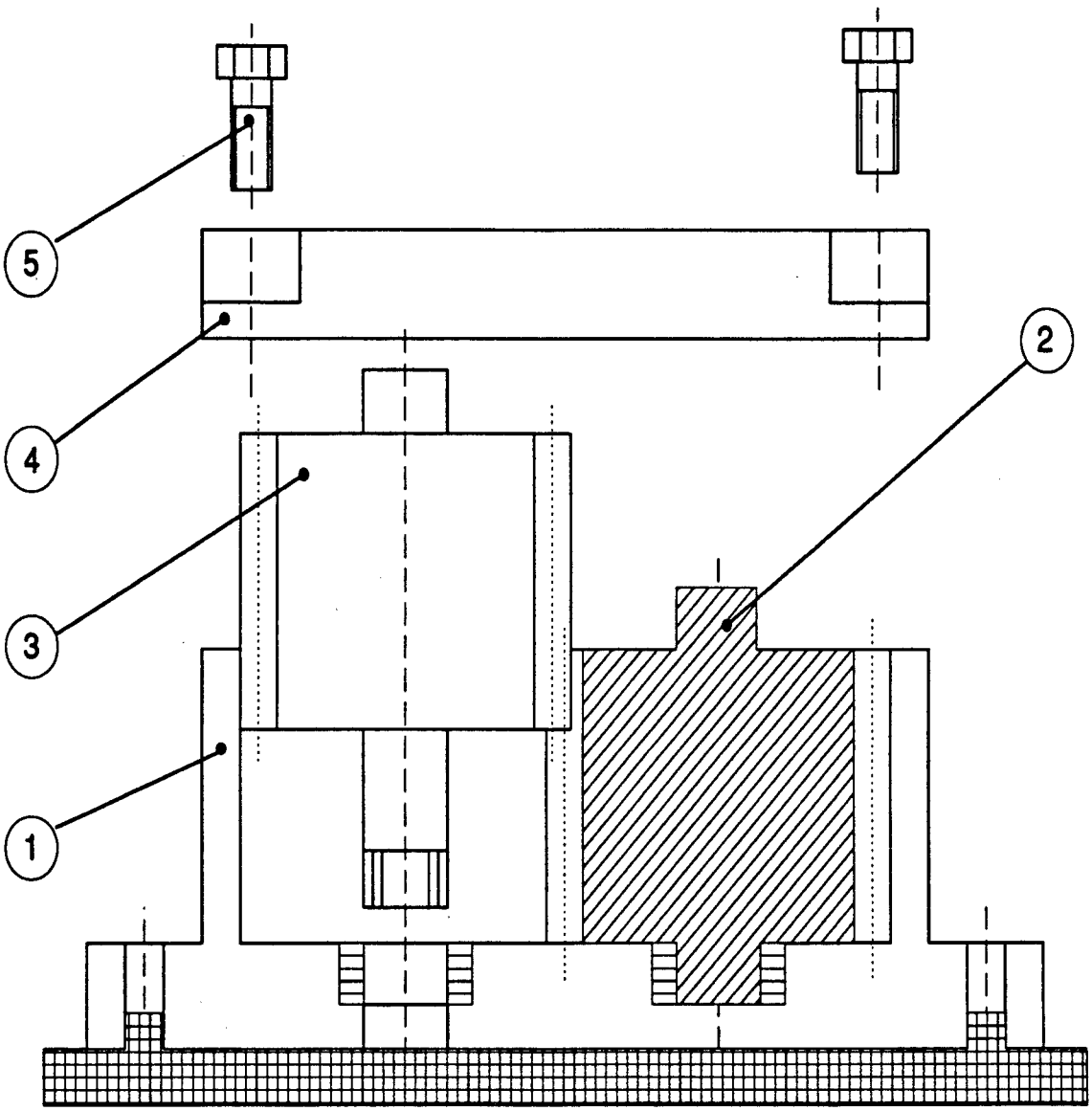

Figure 3.5: Pump assembly parts.

Chapter 4

FORCE/MOTION SYNTHESIS FOR ASSEMBLY TASK

4.1 FORCE CONTROL REVIEW

4.1.1 Introduction:

One of the important problems in executing an assembly task is the kinematic constraint imposed on the motion of the manipulator through contact with the environment. The objective of trajectory control of a constrained manipulator is to regulate force and torque reactions between the end effector (e-e) and the environment while simultaneously regulating desired position and orientation coordinates of the e-e.

The basic control of most of the industrial robots is via position/velocity instructions received from its operator. Robot response depends on forces and torques applied by its actuators and by the contacting environment. In cases like part assembly, a slight misalignment (position error) between the tool's tip and the environment usually causes large damaging forces. This is a classical case for using force control algorithms: the contact forces are used as a source of information on the actual position of the end-effector of the robot relative to the environment. Force feedback offers an important means to enlarge the allowed region of uncertainty, thus avoiding more severe requirements on the positioning accuracy of the robot and peripheral equipment. The problem in practical applications of force controlled (on-the-shelf) robots (Ref. [40]) is that the force controller must always be used in conjunction with a position controller. Most commonly, one wants to specify force control only along selected Cartesian degrees of freedom (D.O.F) while

the remainder are controlled according to a position trajectory (Ref. [21]).

Force feedback strategy is based on information received from a compliant device that deflects due to contact forces that are the result of previous motions. Thus the strategy is governed (Ref. [45]) by the magnitude of the compliances - arm, device and environment, by the dynamic responsiveness of the arm's actuators and by the control computer. Using a stiffness matrix in the forces control algorithms, the force control is reduced to a position control and the accuracy is reduced to the resolution of the position encoders. Ref. [18] concluded that the capability of the robotic system to perform advanced assembly tasks in unstructured and imprecise environment is strongly dependent on its ability to simultaneously control end-effector movements and active forces.

Controlling the movement of the robot arm, while executing an assembly task, has two distinct control phases:

1. gross motion control - an open-loop movement of the arm from its initial location to a (desired) target position/orientation point, along a planned trajectory. In this stage the manipulator performs free (usually high speed) motion without reference environmental contact.

2. fine motion control - a closed-loop motion in which the end-effector interacts with the environment. It is a constrained movement that is carried out using force sensor data.

Robot force control research began in the 1950s, mostlly in the areas of remote manipulators and artificial arm control. In the last decade, work on force control was done at Stanford by Salisbury, Craig and Khatib. Other works were published by Whitney, Mason, Paul, Hogan and many other researchers (like most of the references in this thesis). In Ref. [47], Whitney describes several different block diagrams of force feedback control

algorithms. Fig. 4.1 shows a basic structure for the major part of the force feedback systems. According to this scheme, the robot is controlled by position or velocity commands (input). The desired trajectory is modified by two feedback loops. A position/velocity loop and a force/torque loop.

The common factor to all the force control algorithms (except implicit force feedback) is the use of processed force sensor information in their feedback loops. Algorithms like stiffness control and implicit force feedback use position input, velocity is the input for damping control, while position plus velocity are the inputs for impedance control. Explicit force control use force input while hybrid control uses force plus position/velocity inputs.

Using sensory information (position and force), the robot has to be treated as a sampled data control system. Position trajectory set points are received and acknowledged every 28 (msec), the current position of the internal loops (encoders of the joints) are sampled every 0.875 (msec) and the force sensor information (read by PC program) is received every 28 (msec). The control algorithm must consider the sampling time and the time to process this information to points transformation in order to receive smooth, desired trajectory.

Compliant motion control is concerned with the control of a robot in contact with the environment. There are some approaches to Compliant control, among them:
Impedance control - deals with the desired dynamic relationships between forces and positions. It is simple and robust to parameters uncertainty but, is restricted to fairly slow motions. And *Hybrid control* - the dynamics of the manipulator is calculated in terms of the operational coordinates and then controlling position, force, or mechanical impedance along each axis of the compliance frame.
Both algorithms are discussed in the following sections.

4.1.2 Impedance Control:

Compliant motion of the manipulator occurs when the manipulator position is constrained by the task geometry (contact with the environment). According to Ref. [1], compliance may be derived from the following:

1. Passive Compliance: when force is applied at a point along the manipulator arm, the structure between that point and the base will deflect. Usually the force is applied along the tool held by the gripper, so the deflecting structure includes the tool, the gripper, the sensor and the following elements of the arm: arm links, supporting elements (bearings, guide ways), transmission elements (gears, shafts) and servo drive systems (loop gains in the feedback system). . When the environment is not stiff (related to the other parameters), its stiffness must be added. Due to the fact that nonmetallic materials are involved, the compliance of the arm (plus the other objects) is nonlinear. The nonlinear compliance and the presence of disturbance forces causes inaccuracies of the end effector position.

Oscillations of the arm are deviations of its motion from the programmed motion. The fundamental natural resonance frequencies of bending compliance (only the arm - links and joints) are relatively low (2 to 30 Hz). It mainly effects the performance of high-speed-high-acceleration manipulators. In this work, we will ignore that phenomenon unless it interferes.

Passive mechanical compliance devices like RCC (Remote Center of Compliance) are designed for special-purpose applications (usually in task assembly). Their specifications are known and controlled. They are capable of quick response and are relatively inexpensive (section 2.3.1).

2. Active Compliance: is a programmable device (software algorithm) that enables the operator to modify the end effector's elastic behavior according to different phases of

an assembly task. In a scheme, suggested in Ref. [34], the position gains in a joint based servo system are modified in such a way that the end-effector appears to have a certain stiffness along the Cartesian frame. Describing the stiffness matrix $\mathbf{K}_p$ as the desired elastic behavior needed to generate the force $\mathbf{F}$ due to small displacements $\delta\mathbf{x}$ at the end effector:

$$\mathbf{F} = \mathbf{K}_p \delta\mathbf{x} \tag{4.1}$$

The parameters $\mathbf{F}$, $\mathbf{K}_p$ and $\delta\mathbf{x}$ are expressed in task space coordinate. $\mathbf{K}_p$ is generally chosen to be diagonal with low values along directions which stiffness is controlled and high values along the other directions. Transferring task space forces into joint torques, using the manipulator Jacobian $\delta\mathbf{x} = \mathbf{J}\delta\mathbf{q}$ (expressed in task space), we obtain:

$$\begin{aligned} \tau &= \mathbf{J}^T\mathbf{F} \\ &= (\mathbf{J}^T\mathbf{K}_p\mathbf{J})\delta\mathbf{q} \end{aligned} \tag{4.2}$$

this is an expression for joint torques, necessary to make the hand behave as a six dimensional spring in cartesian (or task) space. The term $\mathbf{J}^T\mathbf{K}_p\mathbf{J}$ is a non diagonal *joint stiffness matrix*. This means that position errors in one joint will affect the commanded torque in all the other joints (the joint stiffness are highly coupled with each other). Eq. 4.2 is a relation between the desired six degrees of freedoms, Cartesian spring stiffness and the controlled joint torques that enables to realize it. Here, through use of the Jacobian, a Cartesian stiffness has been transformed to joint-space stiffness.

Position control in the joint space can be achieved (in the absence of friction) by:

$$\tau = \mathbf{K}_p\mathbf{q} + \mathbf{K}_d\dot{\mathbf{q}} - \mathbf{g}(\mathbf{q}) \tag{4.3}$$

where $\mathbf{q} = \mathbf{q}_d - \mathbf{q}_a$ - the error or the difference between the desired and the actual joint displacement, $\mathbf{g}(\mathbf{q})$ is the estimated gravitational torque and the matrices $\mathbf{K}_p$ and $\mathbf{K}_d$ (programmable damping) are any symmetric matrices.

The *impedance control* is derived using the above development to large task space disc-placements. In this case, joints displacements are replaced with task space displacements vectors ($\mathbf{x} = \mathbf{x}_d - \mathbf{x}_a$) to receive the control law:

$$\tau = \mathbf{J}^T [\mathbf{K}_p \mathbf{x} + \mathbf{K}_d \dot{\mathbf{x}}] - \mathbf{g}(\mathbf{q}) \tag{4.4}$$

here, by expressing $\mathbf{J}^T, \mathbf{K}_p$ and $\mathbf{K}_d$ directly in task space coordinates and give desired values to the "stiffness" and "damping" matrices, the task space force $[\mathbf{K}_p \mathbf{x} + \mathbf{K}_d \dot{\mathbf{x}}]$ is transformed into joint torque vector.

4.1.3 Hybrid Control:

In most of the assembly programs (Subroutines and Service Programs), the arm is constrained by the task geometry. Those programs can be described by using a generalized surface that is defined by position constraints along its normals and force constraints along its tangents (section 3.1). The assembly goals or the desired trajectories can be specified by the operator as another set of artificial constraints, described as position or force constraints. The Hybrid Force/Position Control algorithm was designed to deal with the two different criteria (Ref. [30]) related to the two different types of constraints. The hybrid controller allows the use of force control algorithms along certain degrees of freedom (d.o.f.) while allowing the use of position or damping control algorithms along the remaining d.o.f..

The actuators activate the manipulator's links around their joints while task description and task constraint are declared in the workspace, usually in Cartesian coordinate frame. As in other robot arm control algorithms, significant work has been done by researchers to translate workspace description to joint space. In Ref. [30] both joint positions, compliance, control separation and force were specified in the cartesian coordinate frame. Position errors due to compliance or force received in the cartesian frame

were then translated back to joint coordinate frame using the inverse Jacobian.

A step farther was done (Ref. [28]) by Khatib who resolved the manipulator joint inertia into an effective cartesian inertia. Using the hybrid control scheme, a PID controller was implemented to receive forces (from cartesian accelerations and inertia) and transfer them back into desired torques at the joints. In Ref. [48] the authors modified the hybrid control method by showing that a reduction in the computation loads can be achieved simply by transferring the control from the cartesian frame to joint space. The stability problems of the hybrid control is related to its structure, including both stability problems of the force and the position control algorithms in addition to the implementation of the transitions between them.

Fig. 4.2 shows a typical hybrid force position diagram. The inputs are the desired forces and positions multiplied by the diagonal matrices S and $I - S$. The output of this multiplication is the partition of the forces and positions to the desired six d.o.f. expressed in the cartesian space. Parameters that are received in the cartesian space are transferred to joint space using the Jacobian transpose for forces and inverse Jacobian for positions and velocity.

4.1.4 Force Control for Assembly Task:

The control algorithms described previously can be applied to a robotic arm, by its manufacturer, as part of the software of its controller. Most of the commercial robots can only be activated by using position instructions. In order to apply advanced control algorithms to the arm, researchers like D.G. Bhim (Ref. [3]) designed a new "Universal Six-Joint Controller", capable of driving a robot manipulator equipped with D.C. joint motors and position optical encoders. Such design can directly control the torque of the joints while using the D.C. power amplifiers of a PUMA controller and bypassing completely Unimation controller.

Using built-in functions like SLAVE (Unimation feature that enables the user to control the arm controller by a supervisor computer), can increase the computation ability of the PUMA 560, but it does not allow direct torque control of the joints. Another way to apply force control to a commercial arm is by using the compliance of the arm, while moving the tool in controlled position steps. In this solution, used in this thesis, both position and force control algorithms are used. Activating the arm in an assembly tasks, the same logic as in hybrid control is applied, but here force data is converted to position instructions. So both forces and positions are implemented by using the internal PID control algorithm of Unimation controller.

Every assembly task is divided into the desired position and force controlled d.o.f.. The position control is implemented by using directly the movement instructions of the robot. While in contact with the environment, forces and torques (in the desired d.o.f.) are received through incremental movements. If force is to be increased then a desired, incremental, calculated movement instruction is the output of the force algorithm. The new target point is located in a small segment under the contacting surface (in other words, the robot is instructed to move into the environment). The robot controller responds by calculating (inverse kinematic) some set points. Those are target points used by the inner control loops of the joints to control their movements, using PID control algorithms. As a result, joint actuators will cause the arm (or the environment) to bend until the position error will nullify (joint encoders reaching the desired angles). The result of such pseudo movements will increase forces and torques reactions. Decreasing reactions can be done by entering a target point away from the contacting point.

When the use of the robot includes fast movements, structural dynamic characteristics of the manipulator including stiffness, inertia, damping and natural frequencies are to be considered. Using the arm while moving in contact with the environment involves only fine movements, so part of the dynamic characteristics can be ignored. In this case, force

and torque reactions between the arm and the environment are a function of the step size and the stiffness of the contacting parts. Stiffness parameters that have to be taken into consideration are arm stiffness, stiffness combination of the sensor, gripper and the tool (held by the gripper) and the stiffness of the environment.

Achieving forces/torques and controlling them through the compliance parameter of the system's components can result in high forces, instability and possible damage to e-e components or to the environment. To avoid that problem and to successively control small forces of a stiff arm and environment, the algorithm must instruct the system to move in small steps, resulting slow arm reactions, an impractical way to execute an assembly task. Lowering assembly or arm stiffness by using the RCC and the Linear Compliance Device is the solution that was selected for this problem. In this way, the basic position control of the manipulator (the inner PID algorithm) will still be the high stiffness controlled links required to minimize response to disturbances.

Fig. 4.3 shows the block diagram of the controller activated during assembly task using the following parameters:

1. *task planning:* this is a part of the strategical control level that makes use of task decomposition. Every program, according to its nature, is divided into force and position d.o.f. with their desired velocity parameters. Positions and their associated velocity parameters are transferred directly to the robot controller while the desired forces are calculated using a special control law.

2. *control law:* a force error ($\mathbf{F}_e$) signal is received by subtraction of the sensed force ($\mathbf{F}_s$) from the desired force ($\mathbf{F}_d$). The force error is then processed using the control law shown in Fig. 4.4. The switching control law has a proportional gain ($\mathbf{K}_{fd}$), a dead zone around the desired forces ($\mathbf{F}_e = 0$) and its outputs are bounded to prevent large movements and to receive less nervous control. Usually, the force

part of the control algorithm, is activated using a pc program. The parameters of the control algorithms are derived from the task profile and its nature, except for the dead zone which is usually fixed and receives values that are $\pm 20\%$ of the desired force. For some tasks, when a movement is not needed, increasing of the dead zone is equal to the cancellation of the movement.

3. *robot controller:* the input to the manipulator controller is received directly from the task program (usually a *Subroutine*) with additional real time trajectory alteration from a pc program (usually a *Service Program*). The new desired location is transferred from a' current cartesian or tool frame to joint frame (6 angles) using inner inverse kinematics calculations. Six joints controllers using joint encoders output, close their internal PID control algorithm loops. The resultant movement ($\Delta \mathbf{X}$) of the e-e in one time interval is:

$$\Delta \mathbf{X} = \begin{cases} \dot{\mathbf{X}}_{arm} \cdot \delta t & \text{for } \mathbf{X}_d > \dot{\mathbf{X}}_{arm} \cdot \delta t \\ \mathbf{X}_d & \text{for } \mathbf{X}_d \leq \dot{\mathbf{X}}_{arm} \cdot \delta t \end{cases} \tag{4.5}$$

when

δt - time interval of the system.

$\dot{\mathbf{X}}_{arm}$ - arm velocity (controlled parameter).

$\mathbf{X}_d$ - desired position of the end effector.

In this scheme the robot controller includes the dynamic parameters of both the arm, controller and the electro mechanical subsystems like motors, gears and their amplifiers board.

4. *process and sensor:* the actual forces, locations and velocities of the e-e are function of task specifications, e-e and its equipment and the surroundings environment location and stiffness. The assembly task deals with known parameters (the unknown

are the tolerances of those parameters) so they can be precalculated and feed into the task planner. Actual sensor data is transformed to the desired coordinate frame and feed back to the control law.

Actual implementation of the block diagram and the parameters discussed above are imbedded in the simulation described in chapter 8 and in the programs that executing the assembly task.

4.2 DYNAMIC MODEL ANALYSIS

In ref. [11] the authors are looking to model a robot arm and its sensor in order to receive the conditions for unstable behavior while in contact with the environment. A continuous proportional force controller is introduced in the dynamic description to model an overall behaviour of a position servo system. Fig. 4.6 shows the basic model and the block diagram of its controller. Here, the two mass model includes a rigid robot with mass (m_r) damped to the ground by a damper (d_r). The sensor has stiffness (k_s) and a damper (b_s). The environment adds to the dynamic description, its mass (m_{en}), stiffness (k_{en}) and damping (b_{en}). The model dynamic equations are:

$$\begin{bmatrix} m_r & 0 \\ 0 & m_{en} \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{x}}_r \\ \ddot{\mathbf{x}}_{en} \end{Bmatrix} + \begin{bmatrix} (b_r + b_s) & -b_s \\ -b_s & (b_s + b_{en}) \end{bmatrix} \begin{Bmatrix} \dot{\mathbf{x}}_r \\ \dot{\mathbf{x}}_{en} \end{Bmatrix} + \begin{bmatrix} k_s & -k_s \\ -k_s & (k_s + k_{en}) \end{bmatrix} \begin{Bmatrix} \mathbf{x}_r \\ \mathbf{x}_{en} \end{Bmatrix} = \begin{Bmatrix} \mathbf{F} \\ 0 \end{Bmatrix} \tag{4.6}$$

using the Laplace transformation, the equations can be manipulated to receive the open-loop transfer functions:

$$\begin{aligned} \frac{\mathbf{X}_r(s)}{\mathbf{F}(s)} &= \frac{m_{en}s^2 + (b_{en} + b_s)s + (k_{en} + k_s)}{P(s)} \\ \frac{\mathbf{X}_{en}(s)}{\mathbf{F}(s)} &= \frac{b_s s + k_s}{P(s)} \end{aligned} \tag{4.7}$$

when

$$P(s) = \left[m_r s^2 + (b_r + b_s)s + k_s\right] \times \left[m_{en} s^2 + (b_s + b_{en})s + (k_s + k_{en})\right] - \left[b_s s + k_s r\right]^2$$

these equations give the relationships between the movements and the forces subjected to the system by the motors of the arm. The force information (system's feedback) sensed by the force sensor is:

$$\mathbf{F}_s = k_s(\mathbf{x}_r - \mathbf{x}_{en}) \tag{4.8}$$

using the proportional control law of the form:

$$\mathbf{F} = k_{fd}(\mathbf{F}_d - \mathbf{F}_s) \tag{4.9}$$

The root locus for this simple modeling was calculated and ploted using MATLAB software. Some of the parameters (of the arm) were measured and calculated (chapter 8) and some were estimated (those of the environment). The proportional software gain of the system (K_f) was changed from low values (0.001) to high values (100,000) and the result is ploted in Fig. 4.6. It can be seen that the system is unconditionally stable for any software gain as well as for any stiffness value of the either the sensor or arm.

A more complex structure for the robot arm (that ignors the dynamics of the environment), consist of two masses that represents both rigid body and first vibratory model. The equations for this model are:

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{x}}_1 \\ \ddot{\mathbf{x}}_2 \end{Bmatrix} + \begin{bmatrix} (b_1 + b_2) & -b_2 \\ -b_2 & (b_2 + b_s) \end{bmatrix} \begin{Bmatrix} \dot{\mathbf{x}}_1 \\ \dot{\mathbf{x}}_2 \end{Bmatrix} + \begin{bmatrix} k_2 & -k_2 \\ -k_2 & (k_2 + k_s) \end{bmatrix} \begin{Bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{Bmatrix} = \begin{Bmatrix} \mathbf{F} \\ 0 \end{Bmatrix} \tag{4.10}$$

Once again the Laplace transform is used to obtain the open-loop transfer functions of the equations:

$$\begin{aligned} \frac{\mathbf{X}_1(s)}{\mathbf{F}(s)} &= \frac{m_2 s^2 + (b_e + b_2)s + (k_e + k_2)}{P(s)} \\ \frac{\mathbf{X}_2(s)}{\mathbf{F}(s)} &= \frac{b_2 s + k_2}{P(s)} \end{aligned} \tag{4.11}$$

where

$$\begin{aligned} P(s) = &\left[m_1 s^2 + (b_1 + b_2)s + k_2\right] \times \\ &\left[m_2 s^2 + (b_2 + b_e)s + (k_2 + k_e)\right] - \left[b_2 s + k_2\right]^2 \end{aligned}$$

The contact force is given by

$$\mathbf{F}_e = k_e \mathbf{x}_2 \tag{4.12}$$

and the proportional control law is

$$\mathbf{F} = k_{fd}(\mathbf{F}_d - \mathbf{F}_e) \tag{4.13}$$

In this case, compared with the previous one, it can be seen that the dynamics are the same. The cases only differ in the relative position between the force sensor and the activated force. The root locus output of this case is shown in Fig. 4.8. Once again the parameters were either measured or estimated. It can be seen that in this case the system is only conditionally stable. The actual gain values that cause the instability are not plotted because they depend on the parameters of the system. It can be concluded that when structural compliance is presented between the sensor and the actuator - the controller will attempt to regulate contact forces through the dynamic system and using a certain gains can cause system instability.

In order to investigate the behaviour of the robotic arm while in contact with the environment, a change has to be made in the approach to the system dynamics. Firstly,

the analog controller has to be changed so that a discrete control system can be analyzed (discussed in the following paragraph). Secondly, a simulation has to be implemented using the basic structure of the arm as previously introduced, adding arm stiffness to the dynamic model. This part is included in chapter 8.

In replacing the analog controller by a digital one, a sampler is added between the summing point and the digital controller and a hold (sample and hold) circuit between the digital controller and the plant. This structure produces a time lag (phase lag) that reduces the stability margin of the closed loop system. The transfer function of a zero order hold (see Ref. [17]), may be approximated by:

$$G_h(s) = \frac{1}{\frac{T}{2}s + 1} \tag{4.14}$$

Introducing this hold approximation to a continuous time control system, modifies it to the time lag. The modified analog controller can be discretized and the system can be transferred and analyzed in the z Domain. But, in order to continue the previous analysis, the system has to remain in the s domain. Fig. 4.7 shows the model of the basic, two mass model with first order hold approximation (that model was unconditionally stable using an analog controller). Fig. 4.8 shows the root locus representation of that model. The parameters of the system are the same as in the first, stable case (Fig. 4.5). The root locus shows that the continuous, unconditionally stable system changed to be conditionally stable discrete system.

The scheme that was introduced in this chapter is the basis of the strategy and control algorithms imbedded in the programs that are described in the following chapters. In chapter 8 some reduced parameters of the robot will be added to this model. Using ACSL to simulate the system, the desired spring stiffness (sensor stiffness - $\mathbf{K}_s$) (or software stiffness) will be chosen to achive a guarded move.

4.3 CONCLUSIONS

1. closed loop force control algorithms using stiffness matrix and structured according to Fig. 4.7 are subject to possible stability problems due to time delays or the dynamic response characteristics of the manipulator.

2. applying a discrete controller even to the simple case of the two mass model shows conditional stability. That means that the robot, sensor and environment parameters have to be modeled and simulated. Using the output of that model some parts (stiffness, dampers) have to be designed taking into account model parameters and desired performance.

3. using the measured stiffness of the arm, sampling time of the controller (0.028 [sec]), estimated damping values and low software gains ($\mathbf{K}_{fd} < 1$) in the control law - the robot model is stable (further work on this part is included in chapter 8).

4. both Roberts (Ref. [32]) and Maples (Ref. [21]) come to the conclusion that a mechanically compliant wrist sensor (or compliant environment) permits larger control gains. It has some advantages, due to smaller impact forces (while meeting) and lower contact forces at equilibrium for given approach velocities.

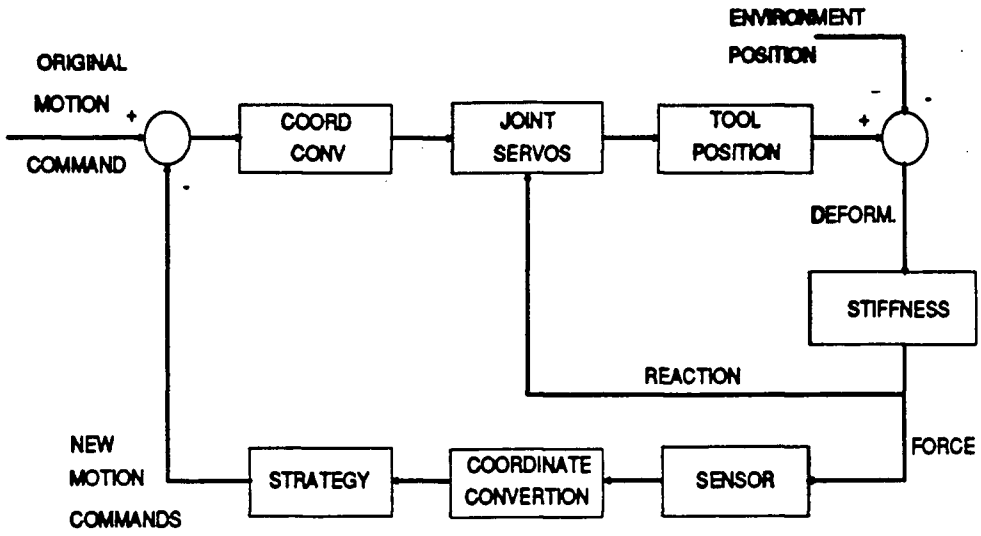

Figure 4.1: Basic force control architecture.

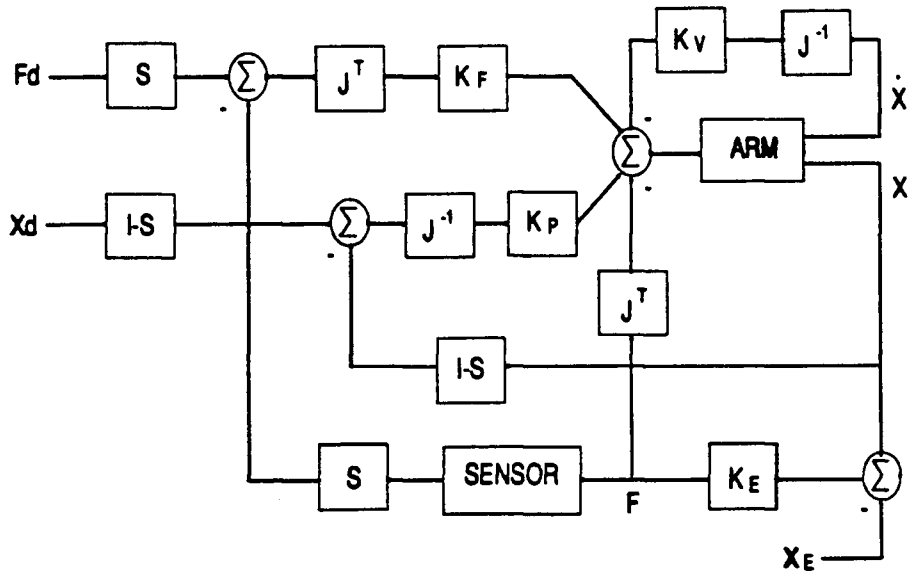

Figure 4.2: Hybrid control block diagram.

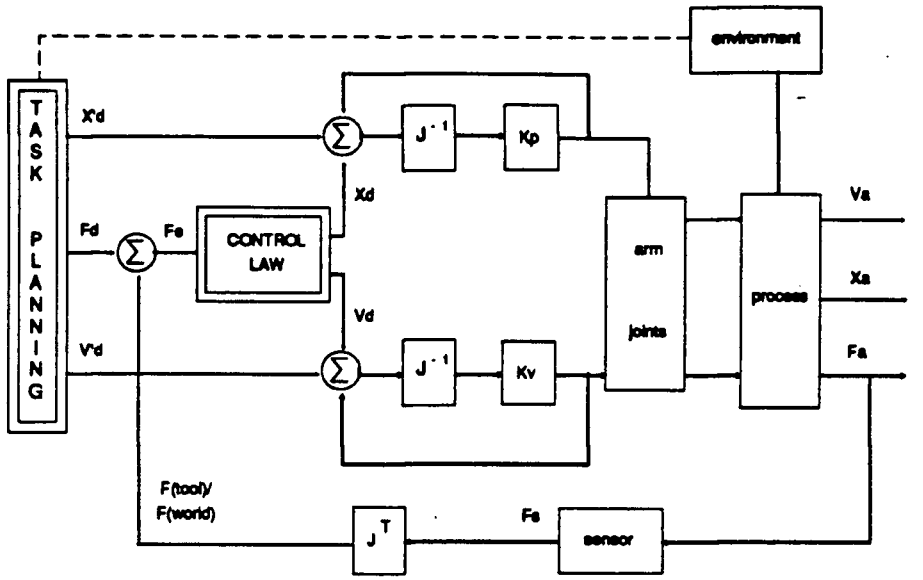

Figure 4.3: Force/position control block diagram.

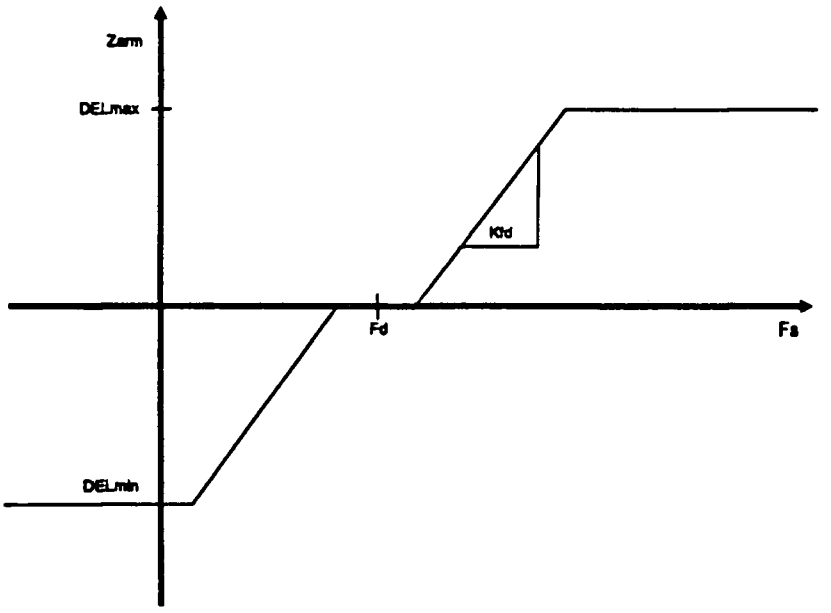

Figure 4.4: Proportional control law.

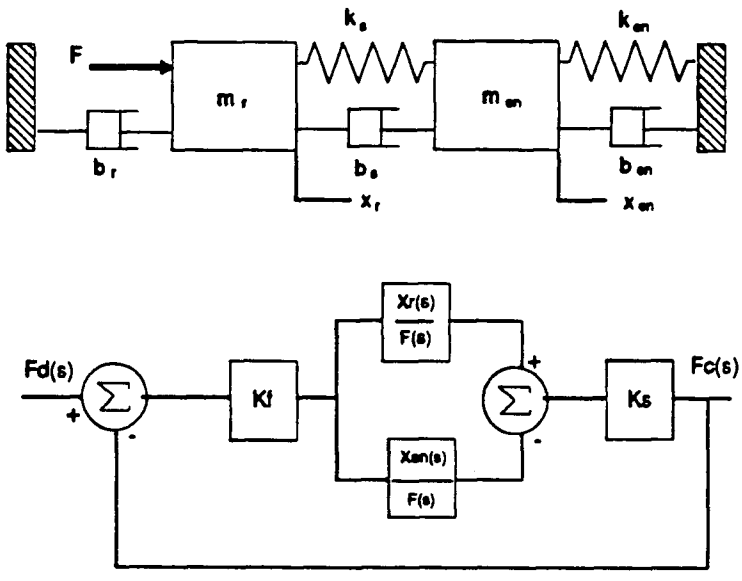

Figure 4.5: Two mass model structure and block diagram.

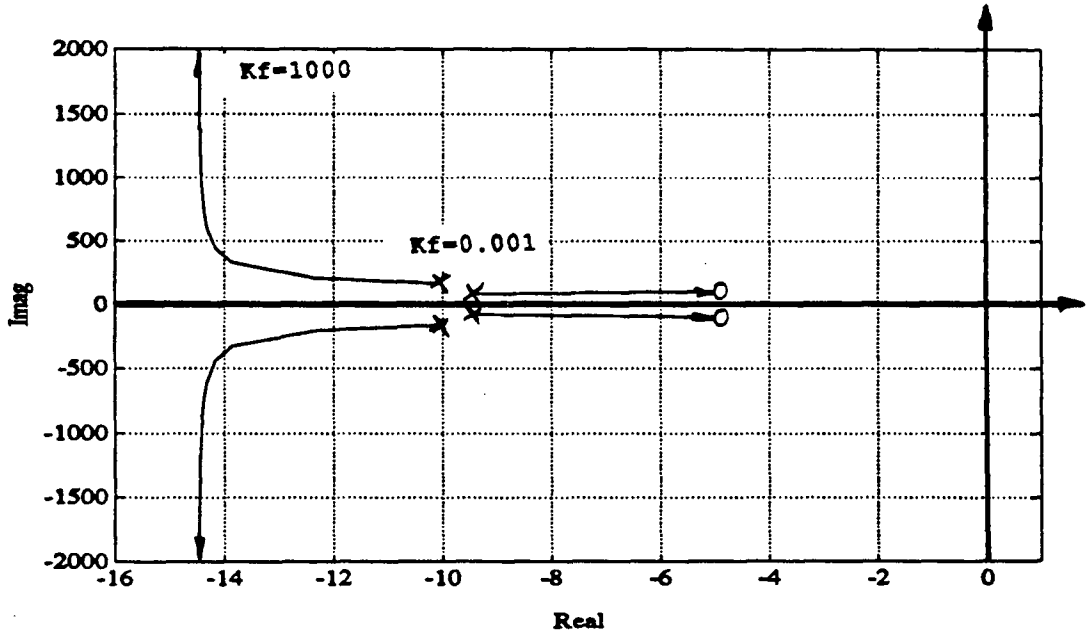

Figure 4.6: Root locus to the model in figure 4.5.

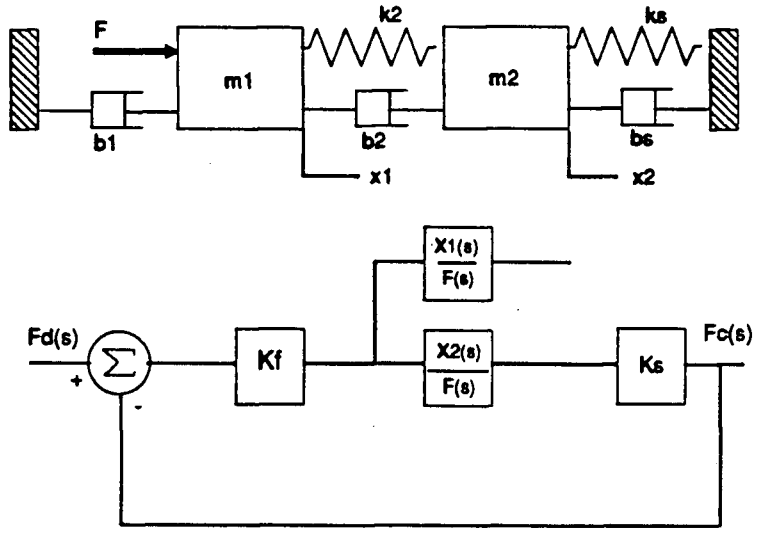

Figure 4.7: Rigid and first vibratory model structure and block diagram.

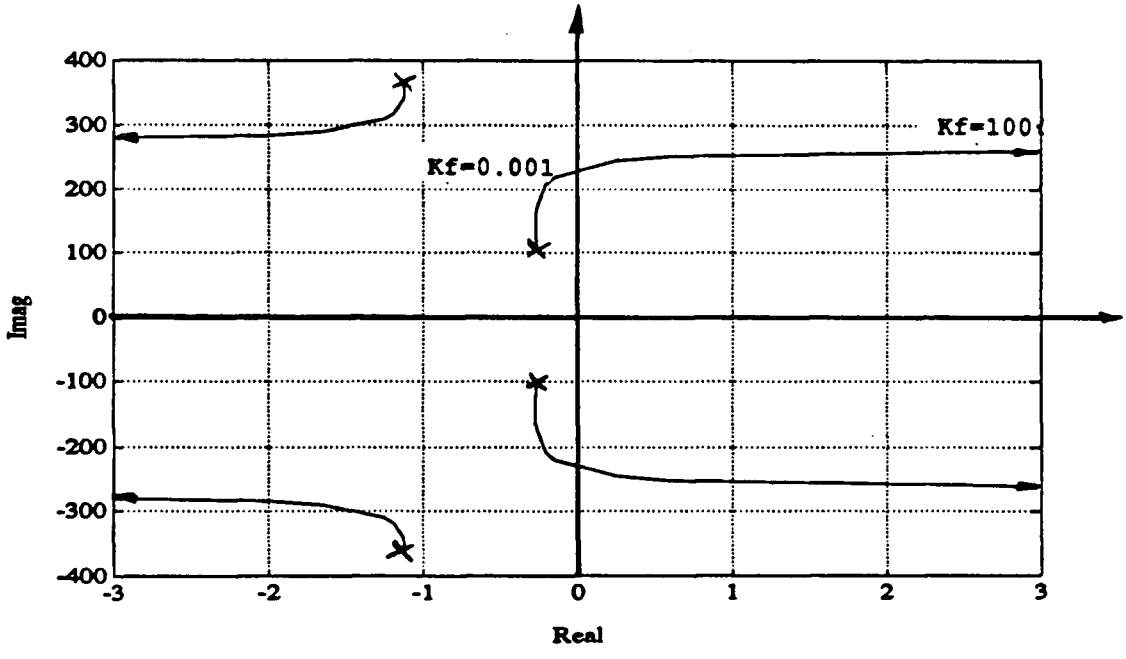

Figure 4.8: Root locus to the model in figure 4.7.

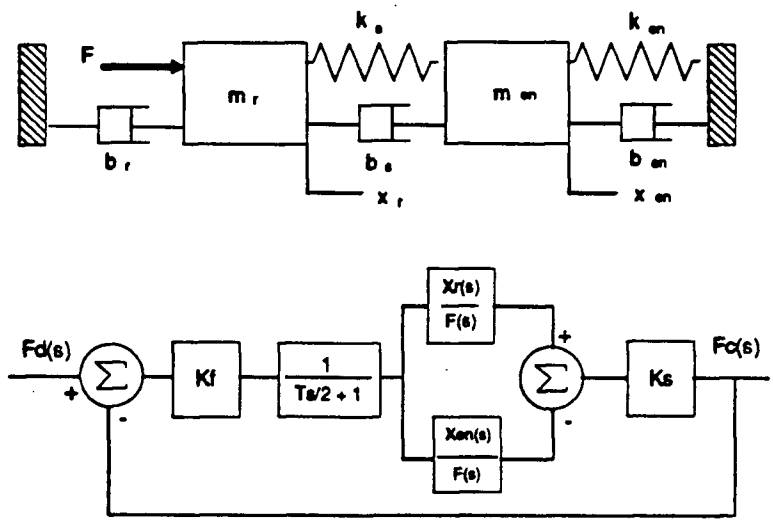

Figure 4.9: Adding zero order hold to two mass model.

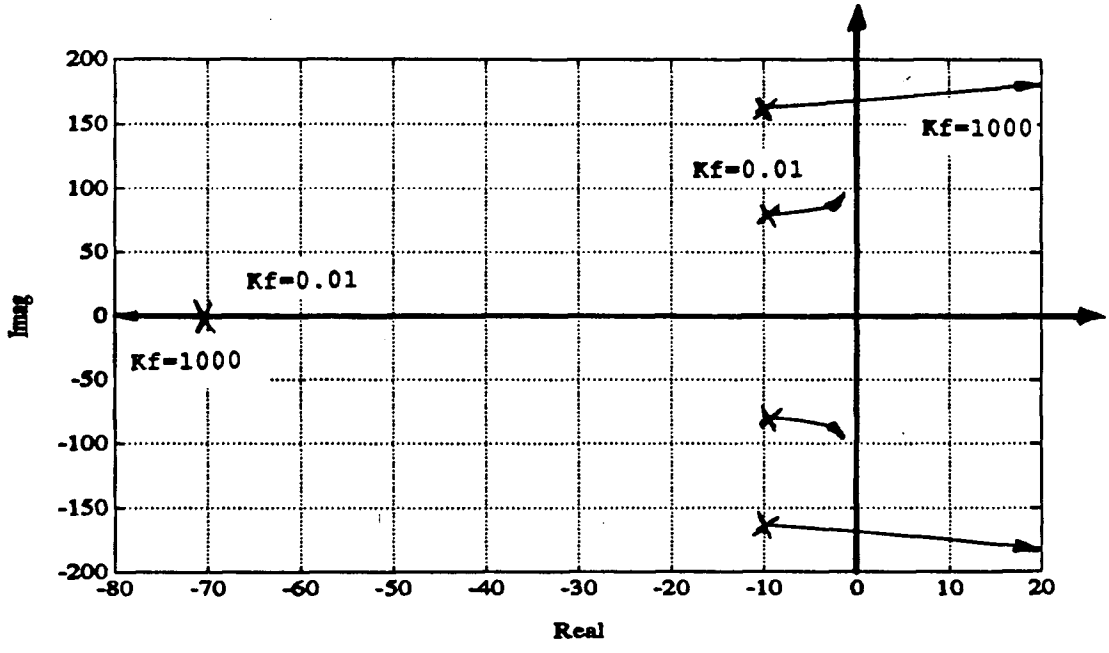

Figure 4.10: Root locus to the discrete model (figure 4.9).

Chapter 5

ASSEMBLY SUBTASKS

An assembly subtask can be a relative operation between the e-e and a tool or a part, such subtasks are **Pick** and **Place**. Another group of subtasks are those that the operation is between the tool or the part and between the environment (workpiece), such subtasks are **Insert** and **Screw**. All subtasks are repeatable operations - they are not oriented to any specific case. In fact, each subtask program contains call instructions that activate subroutines and service programs. According to the logic structure of the assembly library, the main program calls an initialization file before running the desired subtask. Those values of the parameters stay in the robot memory until replaced by new values. So those parameters whose values need no change (during the execution of the main task), can be declared just once in order to save memory space. On the other hand, every subtask has its constant structure including that of the initialization file. It is both easy to program and reduce possible system failure keeping this structure - this is the way chosen for the library programs.

Taking Fig. 5.1 as an example, **Pick** initialization file initializes four subroutines. The parameters of each subroutine are discussed widely in the next chapter, but the fact that all subtasks are using the same subroutines enforced the presence of a list of needed parameters at the end of each subtask section. Those parameters are to be included in the initialization file of each subtask.

As mentioned before, obstacle avoidance and trajectory optimization is not part of this thesis. This problem is avoided simply by declaring a "Free Space" above the workspace

where the e-e with its tool can move freely in the horizontal directions. The straight movement to and from that free space is done by **Approach** and **Clear** subroutines. Every subtask begins and ends in that Free Space.

The subtasks described in the following sections are derived for the special main task - Pump Assembly. Naturally, the library can be used in any other assembly task as well. Thus, it is very likely, that in other assembly tasks, new subtask,subroutines and service programs will be needed. Building those programs, in the logic described here, will increase the library power and flexibility.

5.1 SUBTASK PICK

Subtask **Pick** is designed to pick up a tool or a part from a known target point. **Pick** structure is based on activating four subroutines (Fig. 5.1): **Move**, **Approach**, **Grasp** and **Clear**. **Move** brings the e-e to a point above the target point. This movement is done in the free space above the workspace to avoid obstacle collision. Subroutine **Approach** brings the e-e to contact with the part while **Grasp** subroutine is responsible to grasp it and to confirm the presence of the grasped object - the confirmation is done if a software switch in the initialization file is set to on. Without a vision system, the only way to confirm the grasp is by using the force sensor and weight the object. **Clear** subroutine moves the e-e holding the part back to the free space, ready to make another horizontal movement.

The output of **Pick** subtask is the e-e equipped with a part or a tool. The relative position between the parts is important because forces and moments are activating between them. If it is possible, the designer should avoid to relate on friction as the transformation media for forces and moments. More is given about the subject - in the **Grasp** section.

As shown in the algorithm's structure, the **Pick** initialization file has to initialize four subroutines. The following are the desired parameters that are to be added to the system through every *Pick initialization file:*

1. free space basic height.
2. target point location.
3. movement speeds.
4. end effector configuration (for tool transformation).
5. part sizes from grasping point to tip (tool/force transformation after the grasp).
6. forces to terminate **Approach** subroutine.
7. the followings are special case inputs:
    - max forces to stop **Grasp** subroutine
    - weight and weight trigger=on in cases that grasp verification (by weight) is needed.

5.2 SUBTASK PLACE

Place subtask was created in order to satisfy a very basic movement of a part or a tool from the free space to the target point, using simple position instructions. In some cases, when the target point is not an accurate point, a position plus basic force control algorithms is used in order to verify contact. This option is activated by a software switch.

As all other subtasks, **Place** is built from a list of subroutines, activated by an initialization file. **Place** algorithm structure includes the following subroutines: **Move**,

Approach and **Clear** (Fig. 5.2). The e-e moves horizontally in the free space to a point above the target point, **Approachs** with (or without) force control to the target point, opens the gripper and **Clears** back to the free space.

The following are the desired parameters that are to be added to the system through every *Place initialization file:*

1. free space basic height.
2. target point location.
3. movement speeds.
4. end effector configuration (for tool transformation).
5. part sizes from grasping point to tip (for tool and force transformation after the grasp).
6. forces to terminate **Approach**.

5.3 SUBTASK INSERT

Most assembly tasks commonly use a precise insertion process. Basic examples are tasks such as inserting a pin into a hole or a shaft into a bearing. The insertion task begins while the peg is normal to the hole plane, localized between the hole's boundaries. Execution of insertion tasks is taking place as far as the peg is sliding in the direction of assembly while maintaining the contacts. After the first contact between the mating parts, the insertion task is relatively slow, with small movements. For these reasons, the dynamic parameters of the robot and its loads are ignored.

The usual term for insertion task in the literature is 'peg in hole'. Its repeated use is the reason for receiving fairly large attention in research (mathematical models)

and implementation. Two primary methods were developed for executing these tasks: passive mechanical compliance device and active compliance, implemented in software control loops.

The RCC and the HI-T-HAND are special purpose, passive accommodation devices, designed and built for the peg in hole problem. The basic concepts and the use of the Remote Center of Compliance (RCC) is described in chapter 2. The HI-T-HAND Expert-1 was developed at Hitachi Ltd., Japan (see [14]). It is a tactile controlled robot with flexible wrist and delicate touch which can insert a peg in to a hole with clearance of approximately $20\mu m$. The construction is based on a flexible wrist, equipped with a plate spring. The plate is capable of detecting relative displacements in the cartesian coordinate frame due to the attached strain gauges. The insert operation is performed by correcting and controlling each position and direction, using the sensors data and the flexible wrist.

Active compliance can be implemented using software to change hardware performance. For example, the compliant controlled wrist in reference [7] enables the fine motion of the end-effector. Generally, however, the implementation of active compliance is used as feedback algorithm to control forces and motions of the arm. In controlling a task like peg in hole, both passive control (RCC) and force feedback algorithms are used.

In the following sections, the cylindrical peg in hole insertion will be developed through both a geometrical and force equilibrium viewpoint. The discussion is divided into: ***part mating analysis*** and ***jamming and wedging analysis***. Both sections yield the following results:

- understanding the process in order to derive successful algorithm.
- mathematical representation of the forces and moments involved so constraints and control can be applied to the task.

The last part of the analysis is a summary of the information which leads to insertion strategy and control for successful insertion task.

5.3.1 Part Mating Analysis

The first stage in the insertion process is the mating of the peg and the hole. There are different ways to carry out this process:

a) Reference [39] recommends that during the Approach process, the peg has to be subjected to a certain angular rotation while bringing its lowest point between the hole boundaries. After contact with one flank, the peg is moved to its original orientation and new aligned position. The difficulty of the method is the synchronization between the axes. vanBrussel's experimental system included a wrist, capable of rotation and a jig mounted on an $X - Y$ table (planar demonstration).

b) Reference [46] recommends the use of chamfered holes and summarizes the allowed lateral and angular errors. This basic method is well developed to robot wrist equipped with RCC, while the mating part is mounted on stable jig.

For an assembly process, with manipulator equipped with compliance device, the second method is more efficient (time consuming) and simple. The following subsections gives the forces and moments equations that are to be calculated in order to have force limits and decision points.

The peg is a rigid body, supported by a compliant structure (wrist with or without RCC, robot arm and sometimes the jig of the mating object). The compliance matrix is diagonal at a certain coordinate frame. Placing the compliance center (c.c.) at that specific origin point, simplifies the analyzing and eases the control of the insertion process. In this way the compliant support may be represented as lateral spring constant $\mathbf{K}_x$ and angular spring constant $\mathbf{K}_\theta$. The tip's peg is located a distance L_g from the c.c. (Fig. 5.3). To analyze the forces and moments applied by the support, they are re-expressed

in peg tip coordinates in terms of $\mathbf{F}_x$, $\mathbf{F}_z$ and $\mathbf{M}$ (reference [33]).

During insertion the following phenomena usually occur:

- chamfer crossing.
- one point contact.
- two point contact.

For each phenomenon, the basic information required is the geometric description of the peg and the hole. In order to achieve the successful task, the allowable position errors and extreme forces must be calculated.
The following variables are described as a function of the object's geometry and the compliant of the support:

- U - lateral error of the compliance center.
- θ - rotational error of the compliance center.
- F_z - insertion force.

By controlling the arm via force information from the wrist force sensor and the end-effector movements, all variables are projected to that point.
The section describes the geometric analysis of those phenomena. This information will be combined with the force analysis to achieve a complete description of the insertion task.

Chamfer crossing:

The first part of insertion is the contact between the peg's tip and the face of the chamfer. This part ends when the peg's tip leaves the chamfer and one point contact begins. Fig. 5.4 shows the geometry and forces acting during chamfer crossing.

let: μ = coefficient of friction

$A = cos\alpha + \mu sin\alpha$

$B = sin\alpha - \mu cos\alpha$

The expressions of the lateral and angular errors which lead to the desired insertion force are:

$$U = U_0 + \frac{K_\theta(z/tan\alpha)B}{(K_x L_g^2 + K\theta)B - K_x L_g r A} \tag{5.1}$$

$$\theta = \theta_0 + \frac{K_x(z/tan\alpha)(L_g B - rA}{(K_x L_g^2 + K\theta)B - K_x L_g r A} \tag{5.2}$$

$$\mathbf{F}_z = \frac{K_x K_\theta A(z/tan\alpha)}{(K_x L_g^2 + K\theta)B - K_x L_g r A} \tag{5.3}$$

It must be emphasized that in order to begin the insertion: occur:

$$|\epsilon_0| < w \tag{5.4}$$

Where ϵ_0 is the initial lateral error and w is the width of the chamber.

One point contact:

Fig. 5.5a shows the geometry and forces acting during one point contact, let

$$A = K_x(L_g - l - \mu r) \tag{5.5}$$

where l is the insertion depth and is zero when the peg's tip reaches the bottom of the chamfer. The expressions of the lateral and angular errors which lead to the desired insertion force are:

$$U = U_0 - \frac{K_\theta(\epsilon_0 - R + r + l\theta_0)}{C(L_g - l) + K\theta)} \tag{5.6}$$

$$\theta = \frac{C(\epsilon_0 - R + r + L_g\theta_0) + K_\theta\theta_0}{C(L_g - l) + K\theta} \tag{5.7}$$

$$\mathbf{F}_z = \frac{\mu K_x K_\theta(\epsilon_0 - R + r + l\theta_0)}{C(L_g - l) + K\theta} \tag{5.8}$$

Two point contact:

Fig. 5.5b shows the geometry and forces acting during two point contact.

$$\mathbf{F}_z = \frac{2\mu}{l}[D(\theta_0 - cD/l) + E] + \mu(1 + \mu d/l)[F(\theta_0 - cD/l) - E/L_g] \tag{5.9}$$

where

$$D = K_x L_g^2 + K_\theta \tag{5.10}$$

$$E = K_x L_g(\epsilon_0 + cR) \tag{5.11}$$

$$F = -K_x L_g \tag{5.12}$$

Differentiating $\mathbf{F}_z$ with respect to l, in addition to some reasonable geometrical assumptions yield to a certain point l^* during insertion, where the insertion force $\mathbf{F}_z$ and the contact forces are maximum.

$$l^* = \frac{(4D + 2F\mu d)cD}{2D\theta_0 + E(1 - \mu d/L_g) + F(\theta_0 \mu d - cD)} \tag{5.13}$$

5.3.2 Jamming and Wedging Analysis:

During insertion, two phenomena are acting to prevent the successful completion of the task: jamming and wedging (see [33] and [45]). Jamming is a condition in which the insertion does not proceed in the direction of assembly when a particular force is applied to the peg. It is due to wrong proportions of the forces and moments acting on it. In wedging, no motion is possible in the direction of assembly. This problem occurs due to geometric conditions.

The chapter analyses these two phenomena in order to receive the force restrictions for successful insertion task.

Jamming Analysis:

There are six possible contact patterns between a peg and the edges of a hole (Fig. 5.6). The equilibrium equations which describe the sliding of the peg developed, assuming that the insertion angles are small enough to be ignored.

Combining the equilibrium equations for a peg in two point contact (Fig. 5.5b) and defining $\lambda = l/2r\mu$, force description becomes:

$$\frac{\mathbf{M}}{r\mathbf{F}_z} = \pm\lambda - \frac{\mathbf{F}_x\mu}{\mathbf{F}_z}(\lambda + 1) \tag{5.14}$$

Fig. 5.5a describes equilibrium position for the peg's flank in contact with the hole (one point contact). Force description in this case is:

$$\frac{\mathbf{M}}{r\mathbf{F}_z} = -\lambda - \frac{\mathbf{F}_x\mu}{\mathbf{F}_z}(1 + \lambda) \tag{5.15}$$

Deriving the other possible contact patterns between a peg and a hole are described in the jamming diagram (Fig. 5.6). All the points lie on the two lines:

$$\begin{aligned} \frac{\mathbf{M}}{r\mathbf{F}_z} &= +\lambda - \frac{\mathbf{F}_x\mu}{\mathbf{F}_z}(\lambda + 1) \\ \frac{\mathbf{M}}{r\mathbf{F}_z} &= -\lambda - \frac{\mathbf{F}_x\mu}{\mathbf{F}_z}(\lambda + 1) \end{aligned} \tag{5.16}$$

The jamming parallelogram is defined for certain material combination and lubrication (μ) and for certain geometrical conditions (r,l). The relations between $\mathbf{F}_x$, $\mathbf{F}_z$ and $\mathbf{M}$ defined a point on the diagram. The parallelogram area and boundaries represents sliding-in situation.

The diagram emphasizes the following:

- μ is a parameter that can drastically expand the parallelogram's edges and thou ease the insertion task.

- in order to avoid jamming, the relation between the forces and the moment must be a point within the parallelogram borders. *So, the following equations must exist:*

$$|\frac{\mathbf{F}_x}{\mathbf{F}_z}| < \frac{1}{\mu} \tag{5.17}$$

and

$$|\frac{\mathbf{M}}{r\mathbf{F}_z} + \frac{\mathbf{F}_x\mu}{\mathbf{F}_z}(\lambda + 1)| < \lambda \tag{5.18}$$

Wedging Analysis:

Wedging is a phenomenon dependent on the geometrical relation of the mating parts and on the material combination and lubrication condition (μ). To understand the two point contact phenomenon let us declare:

- at least one of the mating parts is elastic, though still stiff compared with K_x and K_θ.
- the item l is as large as possible, but still allows wedging.
- a cone angle is generated by rotating a line, with angle $\theta = tan^{-1}\mu$, to the normal force. As long as the line of action of the applied force $\mathbf{F}_z$ lies within this cone, there will be no relative motion. This is so, regardless of the magnitude of $\mathbf{F}_z$.

Fig. 5.6b shows a possible position for wedging. The force f_2 at the right side is within the friction cone so there is no relative movement. The reaction force f_1 at the left side points along the lower extreme of the friction cone, indicating that this side is attempting to move out of the hole.

Let l_w be the largest depth at which wedging could still occur, assuming θ and c ($c = 1 - r/R$) are small variables (Fig. 5.3)

$$l\theta = cD$$

$$l_w = \mu d \tag{5.19}$$

assuming $d \approx D$ to receive the smallest angle at which wedging could occur

$$\theta_w = \frac{c}{\mu} \tag{5.20}$$

In order to avoid wedging, the peg's angle must obey

$$|\theta_w| < \frac{c}{\mu} \tag{5.21}$$

or, using the angle description for two point contact

$$\theta_0 + s\epsilon_0 < \pm\frac{c}{\mu} \tag{5.22}$$

where

$$s = \frac{L_g}{L_g^2 + K_\theta/K_x} \tag{5.23}$$

The plot of the geometry constraints on lateral and angular error (while crossing the chamfer) is a parallelogram, received by combining the two restrictions. The allowed errors are those whose solutions are between the boundaries of the parallelogram.

5.3.3 Strategy and Control:

The analysis presented in the last two subsections was carried out in order to achieve tools to ensure successful insertion assembly. The first tool is a correct algorithm based on the understanding of the insertion process. The second tool is the constraints which can be built and controlled between calculated extremes. The following gives the relations between the applied forces, moments and the geometry of the mating parts:

Chamfer crossing: the lateral error of the pin's edge (ϵ_0) must be smaller than the width of the chamfer:

$$|\epsilon_0| < w \tag{5.24}$$

Jamming avoidance:

$$|\frac{\mathbf{M}}{r\mathbf{F}_z} + \frac{\mathbf{F}_x\mu}{\mathbf{F}_z}(\lambda + 1)| < \lambda \tag{5.25}$$

and

$$|\frac{\mathbf{F}_x}{\mathbf{F}_z}| < \frac{1}{\mu} \tag{5.26}$$

The following are the major parameters that are to be added to the system through every *Insert initialization file:*

1. target point location.
2. movement speeds.
3. end effector configuration (for tool transformation).
4. part sizes from grasping point to tip (for tool and force transformation after the grasp).
5. forces to terminate **Approach**.
6. forces, torques, gains, boundary parameters (forces, torques and positions) to control **Comply**.
7. rotation switch on, during the insertion of Gear2 and Top.

5.4 SUBTASK SCREW

An important, typical assembly task is the use of fastening (joining) parts, using bolts, nut, cap screws etc.. In the Pump, four U.N.F., 3/8 hexagonal head, machine screws, fasten the Cover of the pump to its Base. Some design parameters and logic are to be considered while moving those screws from their initial location to their final location on the pump:

- feeding the bolts to their target locations.
- screwing and preloading parameters and the ways to control it.
- tooling.

Subtask **Screw** deals with these parameters for the spacial case of the pump. As usual, it can be expended to any other case by modifying both task parameters and part of the subroutines.

5.4.1 Feeding the Bolts:

For a commercial use, it is efficient to design and operate an automatic screw feeder, that feeds successive bolts to exactly the same loading position. Such a design cancels any position computation, frees memory space and enables the use of the same program to load all the bolts. however in this thesis, to simplify mechanical structures the bolts were placed in an array with equal distances between them. **Pick** subtask is used to pick all the bolts, using relative transformations (from first bolt location) to define the successive shifted location of all the other screws.

One of the solutions for automatic interface between a bolt and its target thread can either be a guidance structure in the screw (shoulders) or a bore before the thread, in the threaded part. It allows guidance of the screw to the beginning of the thread and

an alignment of the center line of the screw with the center line of the thread. In the pump, the threads are located in the base, and the cover has through holes that serves as guidance to the beginning of the thread. Using the holes in the cover, it is easy to solve location problems using **Approach** and **Search** subroutine. At the end of **Pick** subtask, the end effector lives the bolt in the hole, the center lines are quit aligned and it is relatively easy to begin rotating (screwing) the part. The logic applied here to put the bolts in their target location (before the actual screw operation), can be used to the opposite task - to unscrew the bolt and move them back to the parts location.

5.4.2 Screwing and Preloading:

When a body moves in a rotation movement, combined with translation along the same axis, it has an helical motion. When screwing a bolt, the screw, the tool and its driver (in our case - the end effector) has to make an helical motion of rotation and, at the same time, translation along the axis of rotation. The relation between the rotational angle $\Delta\theta$ and the axial translation Δx of a screw with a lead L is

$$\Delta x = \frac{\Delta\theta}{2\Pi} L \tag{5.27}$$

the relation between the axial speed of the screw $V_{ax} = \frac{\Delta x}{\Delta t}$ that is rotated in angular speed $\omega = \frac{\Delta\theta}{\Delta t}$ is

$$\mathbf{V}_{ax} = \omega \frac{L}{2\Pi} \tag{5.28}$$

A possible way to control screw movement is to follow these equations, using both position and velocity instructions. In this case one can calculate the number of turns, required to turn the bolt, to the snug tight condition. From this position, all additional turning develops the useful tension (preloading) in the bolt. For hexagonal structural bolts, like those used in the pump, the turn of the nut (or the screw) should be 180 minimum from the snug tight point, under optimum conditions.

A better approach can be applied to the rotating procedure, using the information received from the force sensor. By studying the behaviour of the torques and forces between the tool and the rotating screw, end effector locations and orientations can be monitored to received just the desired rotating torque. This torque is derived and simplified in reference [36] to be

$$\mathbf{T}_i = 0.20\mathbf{F}_i d \tag{5.29}$$

where

$\mathbf{T}_i$ - the torque required to produce a given preload ($\mathbf{F}_i$).

0.20 - an average value to the torque coefficient (for $\mu = 0.15$).

d - nominal size or basic major diameter of thread.

$\mathbf{F}_i$ - desired preload: $\mathbf{F}_i = 0.90 S_p A_t$.

S_p - proof strength.

A_t - tensile stress area.

Using those equations and the parameters of the 3/8" bolts of the pump, the calculated desired torque is $65[Nm]$. This is the basic, minimum torque to receive preload while ignoring environment influences (fatigue). Such torques cannot be handled directly by the arm or by a tool installed in the gripper (and transfers the torques to the arm). For high torques a special arrangement must be added to the arm, in the thesis, lower torques were used in order to show the ability of the robot to handle complicated tasks. In the next chapter, the description of **Rotate** Subroutine shows the actual way the arm moves using force information to control the screw action.

5.4.3 Tooling:

For the tasks described before, one has to consider how to move the bolts in the workpiece and how to tighten them, using the desired, calculated torques. The fact that the bolts

in the **Pump Assembly** have hexagonal heads made the first task easy. The pneumatic gripper, with 90 groves in its jaws, can grasp the hexagonal head and move the bolts in the workpiece.

As for applying low torques, a square bar, welded to a 3/8" socket, is used to rotate the screws. The bar fits the grooved jaws, and the clearance between the socket and the head of the bolts, lower coupling forces while rotating. A possible problem lays in the fact that while rotating, the contact forces between the square bar and the groves of the gripper tend to open the jaws. For relatively high torques, a small locking, passive device can be added to the square rod that while mounted, enables only small opening of the jaws.

For full preloading, torques must be transferred to the environment. This can be done by using a tool that has a rotary part that rotates the screw and a stationary part, aligned with the environment. In this case, the robot part in the task is to move the tool to its desired locations and activate it through its I/O Module. The positioning of the tool must be done in a spacial way because it has to locate both the screw and the desired location in the environment (double insertion etc.). The following are the major parameters that are to be added to the system through every *Screw initialization file:*

1. target point location and free space location.
2. movement speeds.
3. end effector configuration (for tool transformation).
4. part sizes from grasping point to tip.
5. forces to terminate **Approach**.
6. force and torque, gains and boundary parameters around Z axis.

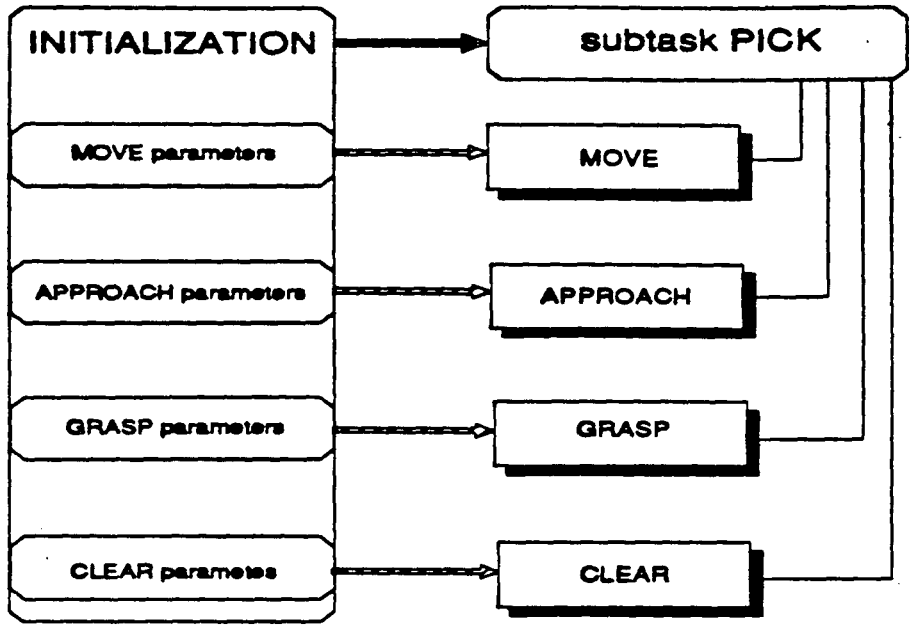

Figure 5.1: Algorithm structure of subtask Pick.

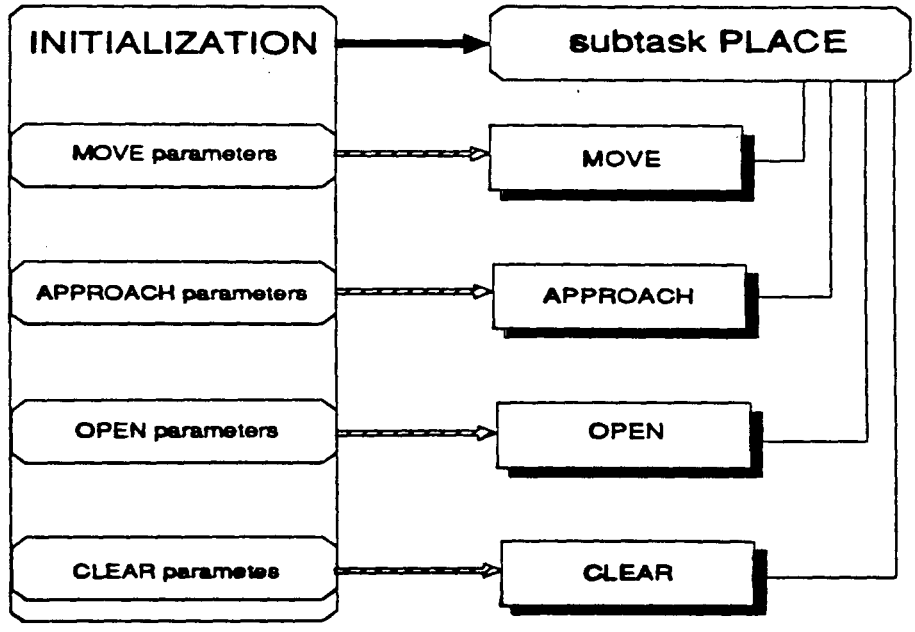

Figure 5.2: Algorithm structure of subtask Place.

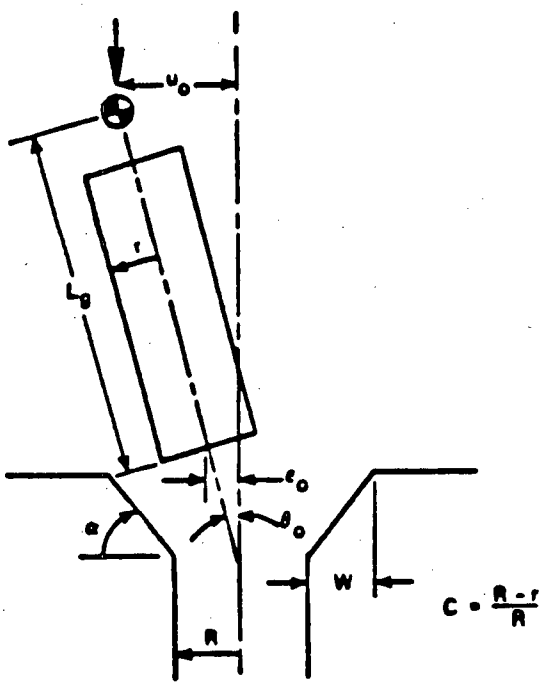

Figure 5.3: Term definition in part mating analysis.

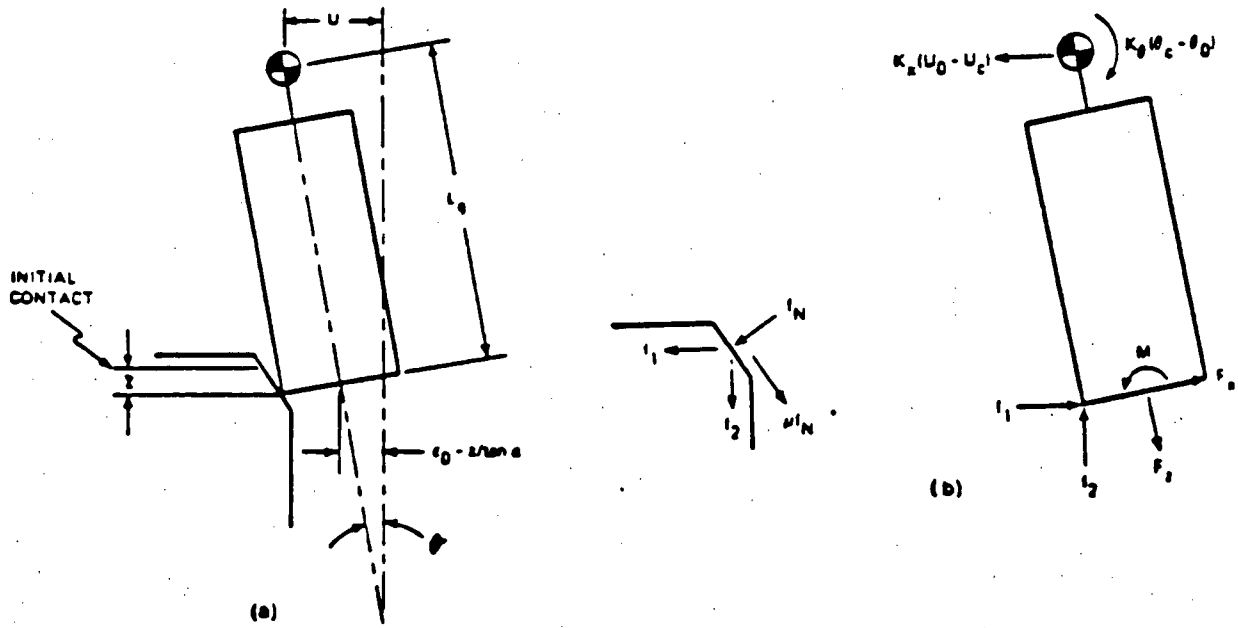

Figure 5.4: Chamfer crossing.

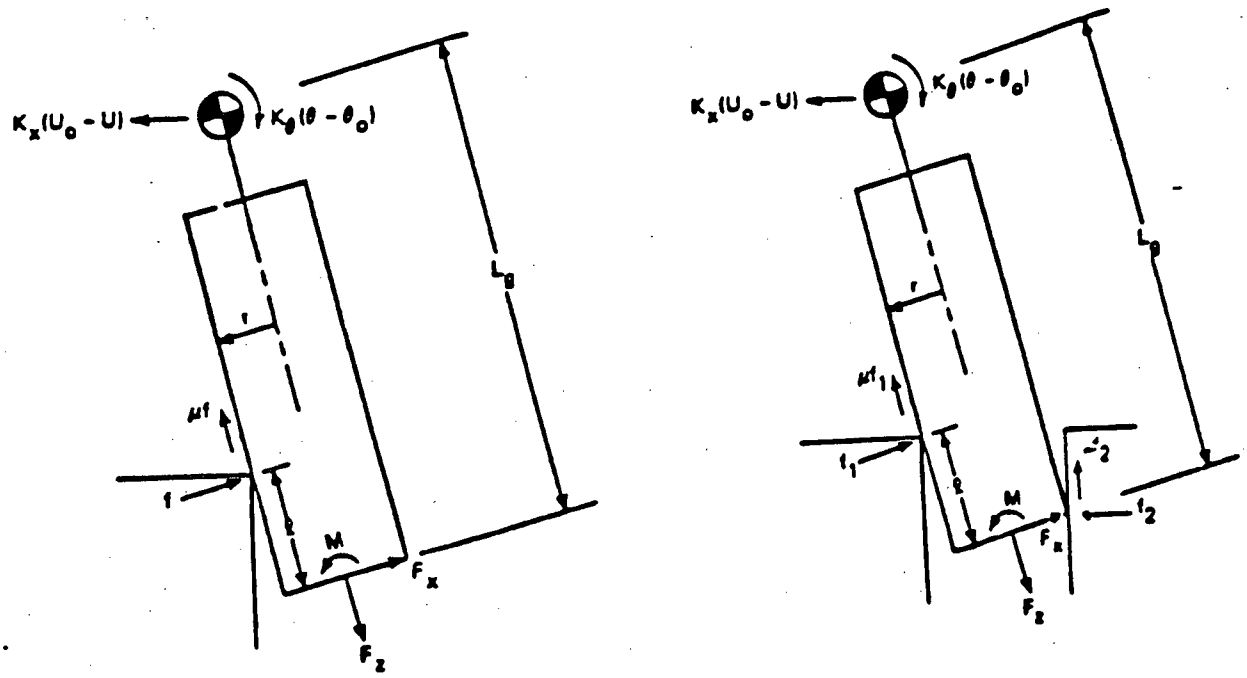

Figure 5.5: One (a) and two (b) point contact.

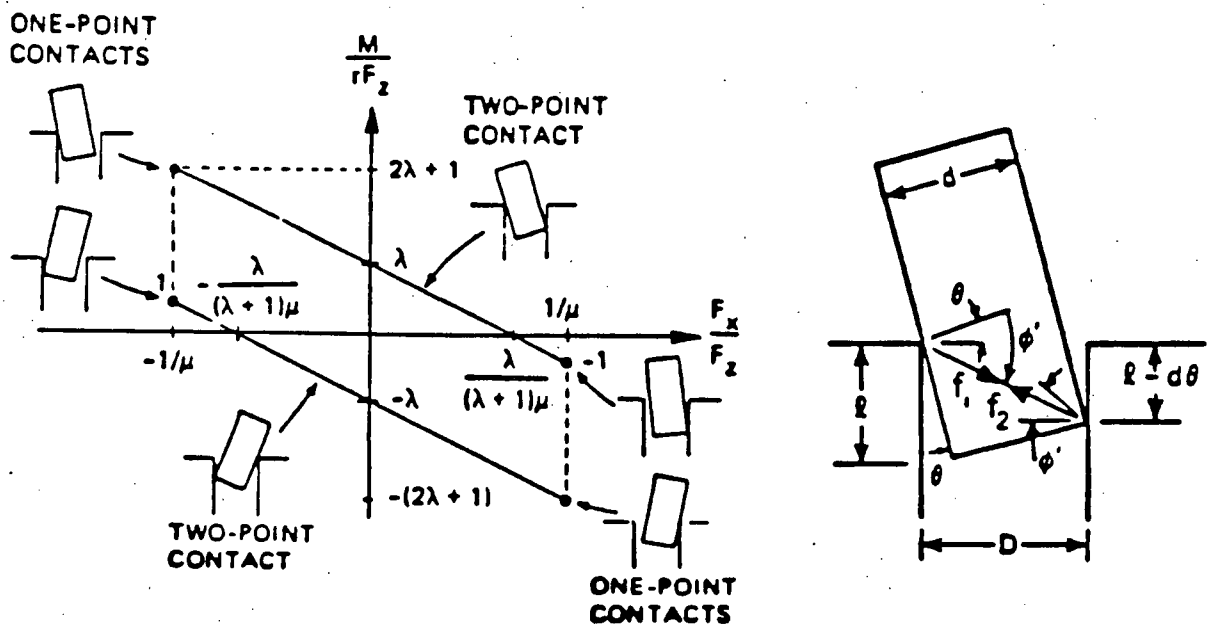

Figure 5.6: The jamming diagram (a) and geometry of wedging condition.

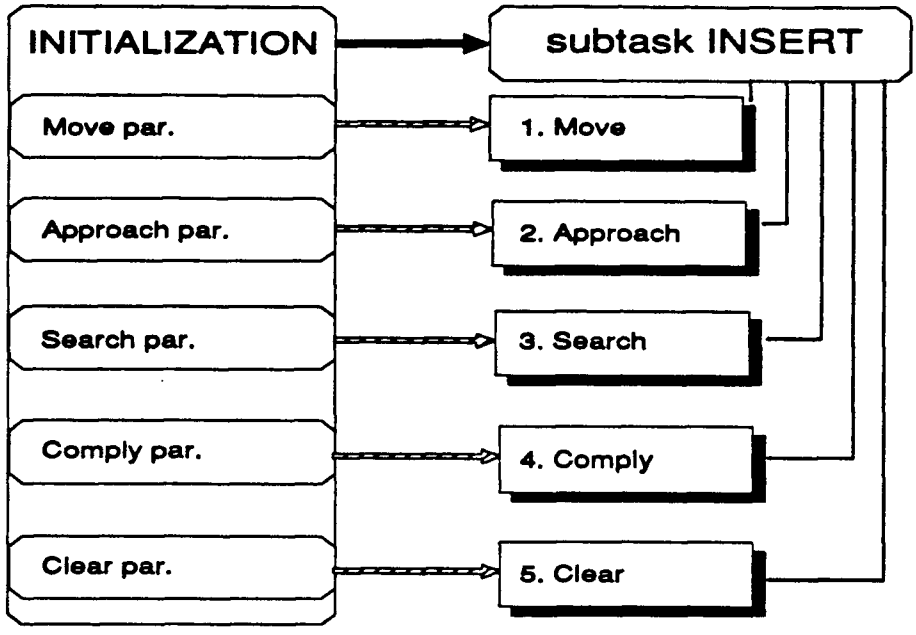

Figure 5.7: Algorithm structure of subtask **Insert**.

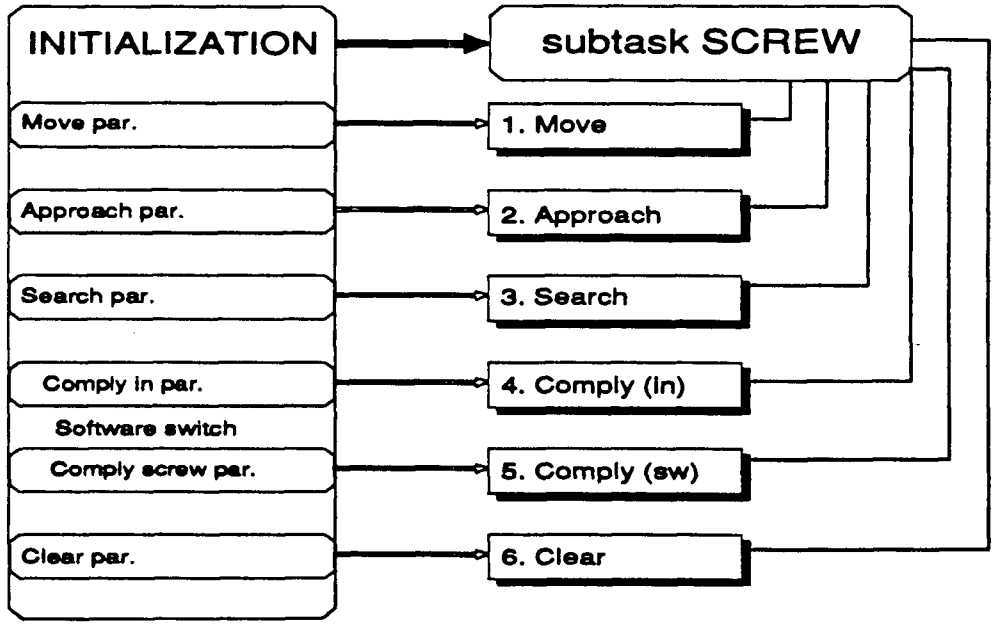

Figure 5.8: Algorithm structure of subtask **Screw**.

Chapter 6

ASSEMBLY SUBROUTINES

During the execution of an assembly task, the subtasks are calling (using call instruction) subroutines that are the repeatable active part in the execution of an assembly task. This chapter describes the principal ways in which those subroutines are carried out, their basic logic structure and the way control algorithms and errors handling are being used. The following subroutines are described in this chapter:

- *Subroutine MOVE:* is used to move the end effector from its current position, to another point, located above the target point, while rotating the tool to a desired orientation.

- *Subroutine APPROACH:* the vertical movement to contact can be done towards a tool or a part in order to grasp them, or, the movement can be done with a tool or a part, in order to contact the environment. Both can be achieved using Approach subroutine with its two stages of movements: 1) rapidly, from a clear point above the workspace to certain height above the contact point. 2) slow, fine motion approach controlled by force feedback loop information.

- *Subroutine GRASP:* to execute an assembly task, parts and tools must be moved from one place to another after being grasped by the gripper of the end effector. Reducing assembly time depends on the flexibility of the gripper and its suitability to task applications. The gripper structure is designed according to the nature of

the task and the configuration of the grasped objects using three principal considerations: safety, reachability and stability.

- *subroutine SEARCH:* after the first contact, the end-effector must obtain a certain angle with the object's plane. Applying a desired search pattern while moving maintains certain contact force (guarded movement).

- *subroutine COMPLY:* causes the tool to react to force information by setting them to the desired values. Forces cause lateral movement and torques cause rotational movements. A force sensor with **Comply** subroutine, acts like an improved (changeable parameters) RCC.

- *subroutine ROTATE:* during screw or unscrew of a bolt, this s subroutine causes the rotation of the tool, while maintaining a constant normal force between the tool and the head of the screw.

- *subroutines CLEAR:* moves the arm back to a point above the workspace or the part location.

Using VAL-II to control the PUMA 560 robot through it's original controller, the only controllable parameters are position and speed of the end effector (or any other point that is declared through "Tool" parameters). Robot locations are used to specify the destinations of robot motions. Two types of locations (both position and orientation) representations are available within VAL-II:

- *precision point:* robot location is represented by the exact position of the individual robot joints. For the PUMA 560, it is defined by the value of angles of the six joints.

- *transformations:* robot location is defined in terms of a cartesian (X,Y,Z) reference frame that is fixed to the base of the robot. The position of the tip of the tool is

defined by X, Y, and Z coordinate and its orientation is defined by three angles measured from those coordinate axis.

Arm movement to a desired location can be executed either by joint interpolated motion or by a movement along a straight line. Real time path control can be implemented only for straight line movements. In this thesis, desired and planned movements are usually activated by subroutines while path modifications are carried out by service programs using real time control algorithms. Those process control programs are activated, and deleted by receiving internal software signals from an appropriate subroutine. Other information is transferred between the pc program and the subroutine using both signals and variable names. This chapter deals with subroutines that were created for the **Pump assembly** task and the next chapter describes their associated pc programs. However, for most of the subroutines, pc programs are unseparated modules that are executing parallel to the subroutine. For this reason, any specific functions related to a specific related to a specific subroutine, is described in the following sections.

Every subroutine is executed from a list of subroutines that is the main part of each Subtask. Each subtask is one point oriented and it manipulate all its subroutines around its specific target point. It is very important that tool transformation (the location of the gripper or the tip of the equipment mounted to the end effector) will be properly introduce to the system before the subroutines are executed. If the tool has one setting while using the "teaching mode" to define the target point transformation, and arm movements are done with another tool setting, correction transformation has to be entered to the system or catastrophic results may occur.

6.1 SUBROUTINE MOVE

The way in which the end effector (with or without a tool or a part) moves, approaches and depart from the assembly area (workspace) or from the part location, can be achieved in many ways. Most of the advanced strategies are based on a model which describes the geometric and physical properties of the robot, the objects and their locations in the workspace. The manipulator trajectory can then be planned according to that model. For this thesis, a free space was located above both the assembly area and the parts location. This volume is used to move the arm freely in horizontal directions using **Move** subroutine. Vertical movements between the free space and the workspace (or the parts area) is done using vertical movements controlled by **Approach** and **Clear** subroutines.

Fig. 6.1 shows the flow chart of subroutine **Move**. The subroutine controls the horizontal movements of the end effector within the free space. It calculates the transformation from the current location to a point above the target point and move the tip of the tool to that new position using certain speed. In order to receive correct movement pattern, the difference between the current Z position (height) and the target Z positions is calculated. If the target location is higher, a vertical movement to that height is executed *before* the horizontal movement. If the target location is lower, the first movement is in the horizontal plane, followed by the vertical movement towards the target point. Using this pattern, horizontal movements are always done in the safe part, defined by the higher point.

If a certain software switch (rotate=1) is active, the movement to the target point is followed by a rotation to a certain angle. The movement to screw a bolt is an example to the need of a such starting position.

Subroutine **Move** (like subroutine **Clear**) is executed when there is no contact between the end effector and the environment. So, no force information is needed and no pc program is activated, the program is carried out, using only the built in position control.

6.2 SUBROUTINE APPROACH

Subroutine **Approach** starting point is when the tool is aligned (perpendicular to the workspace) and its tip is located in the free space a specified distance above the target point. There are two modes in which **Approach** is activated. The first is when the arm approaches (open gripper) a tool or a part in order to grasp it. The second is the approaching with the tool or the part towards contact with another part located in the work area. Every execution of **Approach** subroutine is done towards a pre defined target point. That point can be the grasp point of a tool (part) located in the parts area or it can be on the desired contact point on the mating part located in the workspace.

Approaching to grasp: this movement will be usually executed towards a point located in the parts location in order to grasp a part or a tool. When $open.grip = 1$ and $weight.ex = 1$ are declared in the initialization program, Weight service program will weight the end effector configuration before it start to move. The gripper is then opened and a rapid motion begins towards the shift point (previously calculated using the target point as reference). The rapid motion is not force controlled because system reactions to overloads are faster then those of the arm due to force control so if something does happen, arm power will shout down. The movement from the shift point to the target point is done slowly and the program is terminated when it reaches the target point.

Approaching to contact: this approach mode is activated when the gripper is equipped with a part or a tool and the movement is towards the contact with an assembly part or towards the parts location, to return a tool. As before, the first movement towards

the shifting point is done rapidly (close gripper, no weighting). At the shift point, **Approach.pc** is activated and real force information is transferred to the main program. Here, no path modifications are needed and a pc program was activated in order to quick program response. The movement towards the target point is done while checking force data to confirm contact. The movement can be terminated (control return to the subtask) either by reaching a certain maximum force or certain lower position. If those contact forces are detected before the tip of the tool reaches target point location, the program will be terminated. If movement continues (maximum force are not detected) until the tool reaches certain lower point beyond the target point (min.z), the program is terminated and a software switch will indicate that **Search** subroutine should not be executed (search.ex=0).

For some applications, its is reasonable (according to task description) that the end effector will face forces *before* the desired point is reached. The contact between the tool and the bolt's head is an example when a special small rotation is activated during the approaching pattern (activated by a software switch). This enables contact forces to end the approach movement, according to force information received from a contact that occurs while the screw's head is inside the tool (desired location) and not outside it (possible first contact).

Approach subroutine and its pc program Approach.pc flow charts are shown in Fig. 6.2. All the parameters needed to properly execute those programs (tool transformation, locations, speeds and force limitations) are basically defined in **Approach.par** and are to be properly changed in the appropriate initialization file.

6.3 SUBROUTINE GRASP

6.3.1 Grasp Planning:

In order to execute an assembly task, the *target object* (parts or tools to be grasped) must be grasped and moved to a certain point, in a certain path. The movement and the proceeding operations are deeply influenced by the shapes of the gripper and the target object and by the choices made during grasping. In Ref. [22] the authors designed a multi-functional gripper that can handle all the actions desired during task execution. Based on coasts calculation and productivity, they concluded that for minimum assembly time (and costs), in complex applications, the gripper requires the greatest attention.

Grasp configuration is usually based on the following:

- *Reachability:* the robot must be able to reach the object without interference. Then, to move to its target point in a collision free path, while holding the target object.
- *Safety:* during approaching (with or without the object) and moving procedures, the robot must be safe.
- *Stability:* the grasp should be stable due to forces acting on the target object during transfer motions and parts mating operations.

An important issue in controlling the grasp operation is the verification that the target object was grasped correctly. This means that the target object was successfully grasped between the gripper's jaws and its position and orientation are as planned. Checking this information can be done easily by using a vision system or a tactile gripper. A force sensor (like the one in CAMROL) can be used to weight the object after grasping (Weight subroutine). Using this subroutine can verify that the object is held by the gripper's jaws, but it tells nothing about *how* it is being held, that is an important input for the following steps.

In order to by pass this problem, a servo-controlled gripper can be used along with the force sensor. Alternatively, by working with the force sensor alone, the movement has to consider the part, the gripper and the workspace. An example can be taken from the way the gears are being grasp in Pump assembly task. **Approach** movement is ended in a point along the gear's shaft, **Grasp** continues this movement in a guarded move and apply sliding motion on the shaft until the gripper hits the structure of the tooth. This ensures fixed relative position between the tip of the gear and the tool's coordinate frame. For other parts, other logic is applied - all in order to align the grasped part with the pre-defined location of its tip.

If there is no possibility to verify the relative location of the grasped object during the grasp action, the object can be moved to contact (force controlled) with the edges of a cube. Using force information, compliance data of the arm and the grasped object and the relative location of the cube's edges, the location of the tip of the object can be calculated.

6.3.2 Gripper's Configuration:

One of the dominant principles in grasp planning is the interaction of the gripper's shape and that of the target object. In planning the assembly task, the designer must choose a gripper's shape that will match the target objects and its function in the whole task. Another constraint on the configuration of the gripper is its interaction with the environment. According to [4], the candidate grasp configurations are those having the gripping surfaces in contact with the target object while avoiding collisions between the manipulator and other objects. Current proposals for grasp planning assume a limited class of object models: polyhedra and cylinders. The popular, simple type of gripper is a parallel-jaw gripper. A parallel-jaw gripper, pneumatic actuated, is used in CAMROL.

Many of the target objects are screws, pins, screw driver heads (allen keys) etc.

While grasping those objects the gripper must align their Z axis with its own Z axis. The designer of the gripper should take the following problems into consideration:

parallelism of the jaws: usually, parallel tracks at the base of the gripper cause the parallel movement of the jaws. Extensively used jaws which were subjected to high forces and moments may no longer be parallel and cannot align with the target tool while grasping it. Using short jaws with 3 designed contact points (2 in one jaw and 1 in the other) can improve the problem considerably.

relative position along the Z axis: moving in guarded motion or while parts mating occurs, the tangential forces acting up on the target object from the environment are restricted by UNIMATION to 58 [N]. Assuming static friction coefficient of 0.5, the normal forces supplied by the jaws must be 116 [N]. However, with the current system this cannot be achieved, unless a mechanical stopper is placed in an aligned position or, the tip of the gripper is pressed to a groove in the part.

torque transmission: applying torques on bolts heads or any other shape that is not rounded, applies normal opening forces on the gripper's jaws. For application that the forces are higher than the jaws's closing force, either the tool (screwdriver), or the gripper, must have mechanical stopers that prevent the opening of the jaws while rotating.

Grasp subroutine and its pc program **Grasp.pc** flow charts are shown in Fig. 6.3. All the parameters needed to execute properly those programs (software switches, tool transformation, locations, speeds and force limitations) are basically defined in **Grasp.par**.

6.4 SUBROUTINE SEARCH

One of the modes of subroutine **Approach** is the approaching to contact. The gripper is equipped with a part or tool that is moved towards the environment for first part mating. The last movement is done slowly to a pre defined contact point. At this last

stage, either the normal forces will increased, indicating that part mating had occurred or, force information will not changed. If the forces are not changing while the tool's tip passes certain location, either the desired point was not found (the assembly part is not in its place) or, an insertion process has began. If subroutine **Approach** is ended because normal forces were detected (in a point near the location of the target point), then the active Insert subtask will call subroutine **Search** to locate the desired insertion point.

6.4.1 Search strategy:

Search subroutine was created in order to detect points that cause change in the force information when contacting the tip of the tool. Such points are hole edges, part borders, walls etc.. The actual pattern can be divided into two basic movements:

- full square search pattern, when the movements are in the X-Y plane, along the X and Y coordinate frame. This pattern is usually used when the desired location was not found and it is assumed that the location of the hole is near the tip of the tool.

- a search pattern along one axis in the X-Y plane. Usually this search is the complementary task to the full square search, when one edge was located the search continue in one direction in order to find the second edge.

Fig. 6.4 shows the square pattern of the search movements in the X-Y plane. Every second change in the direction of the movement, increases the size of square's ribs. The movement along each rib is done by using an internal do loop that breaks the movement into a series of small steps. After each step, force signals from the pc program are checked, so the main program can easily react and stop the movement at that certain point, where the possible edge was found. The square search pattern is then changes into

a movement, perpendicular to the detected force. This axis search pattern uses the point where the force was detected as a new Ref. point, and the steps are around that point. The program ends when a desired force in the second axis is detected.

During program execution, it is important to have certain points that are related to the workspace. The beginning point of the program is the defined, target point of subroutine **Approach**. That point is used to calculate temporary target points, located at the end of each rib during arm movement in the square pattern. Arm movements around that point can be done easily, using SHIFT instruction while the number of steps is increased and the sign (movement direction) is changed. However, it was found, that when a lot of successive transformations are executed (one related to the other), the arm will not reach the location target. It may be a result of a cumulative error, created by the controller while calculating a large amount of transformations of small numbers. The error causes relatively high arbitrary shift in arm location from the desired, designed trajectory and target points.

From the structure of the square search pattern, it can be seen that in order to continue four full movements, x and y are moved twice to a position located $-\delta x$ and $-\delta y$ from the starting point (ini.pnt). After another full movement the final position is located in a distance that is $-2 * deltax$ and $-2 * \delta y$ from ini.pnt and so on. Around those locations, the algorithm moves the end effector, using small, force controlled steps. In this way, even if there is a shift while the arm is moved (using relative transformation) in any rib of the square, it will recover due the use of a known location in the followed step.

During the movement, directions are changed (sign of the movement), the number of cycles are increased and so are the number of steps used to reached the end of each rib. Fig. 6.5 shows the parameters and the way they are used to achieved a complete search pattern (using VAL-II instructions). When the square pattern is changed to a one axis

movement, those parameters are null, and the movement in the direction of the detected force is canceled. The movement if the direction, perpendicular to the detected force, is executed using same scheme as was used for the square pattern.

While the main program uses position control to moves the arm in the X-Y plane, the pc program (**Search.pc**) is used to maintain contact with the surface, using force control algorithm that alters the movement in the Z direction. **Search.pc** is also used to send forces signals to the main program, indicating edges and contact forces. This kind of movement that follows the shape of the part in contact is called - guarded move. There are two ways to control the contact forces with the environment:

- to use high gains between the force information and the movement in the Z direction so the arm will response quickly to any change in the environment location (can not be implemented without addition compliance device).

- to use sensory information about the contact forces. If they do not remain between desired boundaries, the movements in the X-Y direction is stopped, a movement from or toward the environment is performed until the contact forces receive their desired values. Only then the X-Y movement continue.

the first mode is quicker, sensitive and nervous. Small jumps will happen if the guarded move follows a sloped contour. The second way is smooth but slow, it will be used for very small motions, while working with closed fitted parts. both modes can be used with the basic algorithm, for the pump - the first mode is used.

In some special cases, a successive insertion has to be performed after the first insertion. In this case, a certain region of the part has to perform search pattern, while another region is constrained and can not be moved in the X-Y plane. Using the constrained region as rotating plane, the search pattern is performed using rotational movement instead of the lateral movements described before. This procedure make use of rotation around X

and around Y while the pc program enables guarded move, using tool coordinate frame. Fig. 6.6 shows a possible situation that occurs while inserting the gears to their locations in the pump.

Subroutine **Search** (Fig. 6.7) is a good example where a compliant device, installed in the Z direction, can make the program and the process simple, reliable and much quicker. With the high stiffness of the arm, only small steps can be used to move the arm into the environment and slow movement must be used in the X-Y directions. Otherwise, when small tolerances are presented, movements in the Z direction will not be able to constantly trace the surface, and the part will miss the edges of the hole (this part is discussed and checked in chapter 8 by simulating a guarded move).

6.4.2 Rotation strategy:

There are many cases (Ref. [40]) where an easy detection of the hole can be done after subjecting the tool (part) to an angular rotation.

- rotate the tool to a certain angle (usually 70) with the X-Y plane.
- bring the front edge of the tool between the hole's boundaries. This is done by pointing with the tip of the edge to the direction of the target point.
- apply full search pattern until edges forces are received (that means that there is contact between the edge of the tool and the edge of the hole.
- while maintaining contact, rotate the tool back to be perpendicular to the surface of the environment.

To use this mode, tool parameters has to be carefully define and the rotation must be done using tool coordinate frame. In the thesis, the rotation part was used in the assembly of the Top of the pump.

6.5 SUBROUTINE COMPLY

All the program that are using the force sensor data are built to react to feedback information by analysing it and moving the arm according to the program logic and the control law. In many cases where the arm has to follow a contour (even an insert task is done by sensing, reacting and in a way, following the hole's contour), good results may be achieved by complying tool movements to the forces and torques acting on the contact point. In this case, the robot will react to forces and torques that exceed certain envelope, by moving to certain directions that will drive the forces and torques values to be inside the desired envelope.

The next chapter deals with the ways to transfer force data to any desired location in any angle. This allow the program to resolve force information to its components and to rotate it to the same coordinate frame that is used to control the movement of the arm. Force information is translated to the contact point location (approximately known), using tool transformation and arm movements are controlled according to that point. By manipulating forces and torques to the same location and coordinate frame that is used to control the movement of the tool, each degree of freedom can be handled alone. It is important particularly when arm control is applied by putting special constraints on one (or more) d.o.f. and it is also ease task programing and assembly debugging.

In a way, all the control programs in the thesis are acting on the same base. For example: **Approach** subroutine moves the arm to contact and reacts to small forces by continue the movement in the Z direction towards contact. Once a contact was reached, the program is terminated. Usually, **Comply** is activated using all the six degrees of freedom. Each d.o.f. has the same force/position control structure and the same proportional force control law. But each d.o.f. has other initial conditions that defines the borders of its force envelope, direction of movements and gains.

Comply subroutine and its pc program **Comply.pc** flow charts are shown in Fig. 6.8. All the parameters needed to execute properly those programs (control low borders and gains, tool transformation, locations and force limitations) are basically defined in **Comply.par**.

6.6 SUBROUTINE ROTATE

Subroutine **Rotate** is the actual part in subtask Screw that rotates the screw in either clockwise or counter clockwise directions. The very original designed structure of rotating the screw was to use **Comply** subroutine with the certain set point that will cause arm rotation around the sixth d.o.f. (rotation about Z axis). This can be done by using a DELAY instruction in the main program. Delay instructs the robot to move to nowhere and enables the pc program to alter and rotate the arm according to force information. Here, instead of complying all the six d.o.f., the tool is rotated until certain, desired torque is reached.

Here again, as in **Search** subroutine, using the pc program to rotate the end effector causes the calculation of large number of transformations. This is done with one transformation follows the previous one and the result is a shift in the tool's location and instead of pure rotation, an arbitrary movement is executed that causes undesired forces and torques between the tool and the screw.

Using the structure showed in Fig. 6.9, the rotation is done in the main program, using the starting point as reference location. The pc program (**Rotate.pc**) detect the torque and if it reach certain level, an inner signal causes the end of the program. The pc program alters arm's movement only in the Z direction when a constant normal force is needed. As in the other cases, **Rotate** parameters are listed in the file **Rotate.par**.

Rotate subroutine shows the way and the ability of the end effector, but actually,

the screw is not rotated to the desired torques, calculated according to the equations derived in chapter 5. For a 3/8" bolt, the minimum desired torque to receive preload is 65[Nm] (ignoring fatigue loading calculations). Such torques cannot be handled directly by the arm and so, are not applied using this subroutine.

6.7 SUBROUTINE CLEAR

Arm movements is done using locations and speeds combined with Move instructions.

Subroutine **Clear** makes the opposite movements of subroutine **Approach**, it moves the arm from contact point to a point located in the free space. The subroutine is used to depart with or without a tool or a part. This information is received from the initialization file using a software switch.

Fig. 6.10 shows the flow chart of subroutine **Clear**. The subroutine controls arm movement using locations and speed, combined with MOVE instructions.

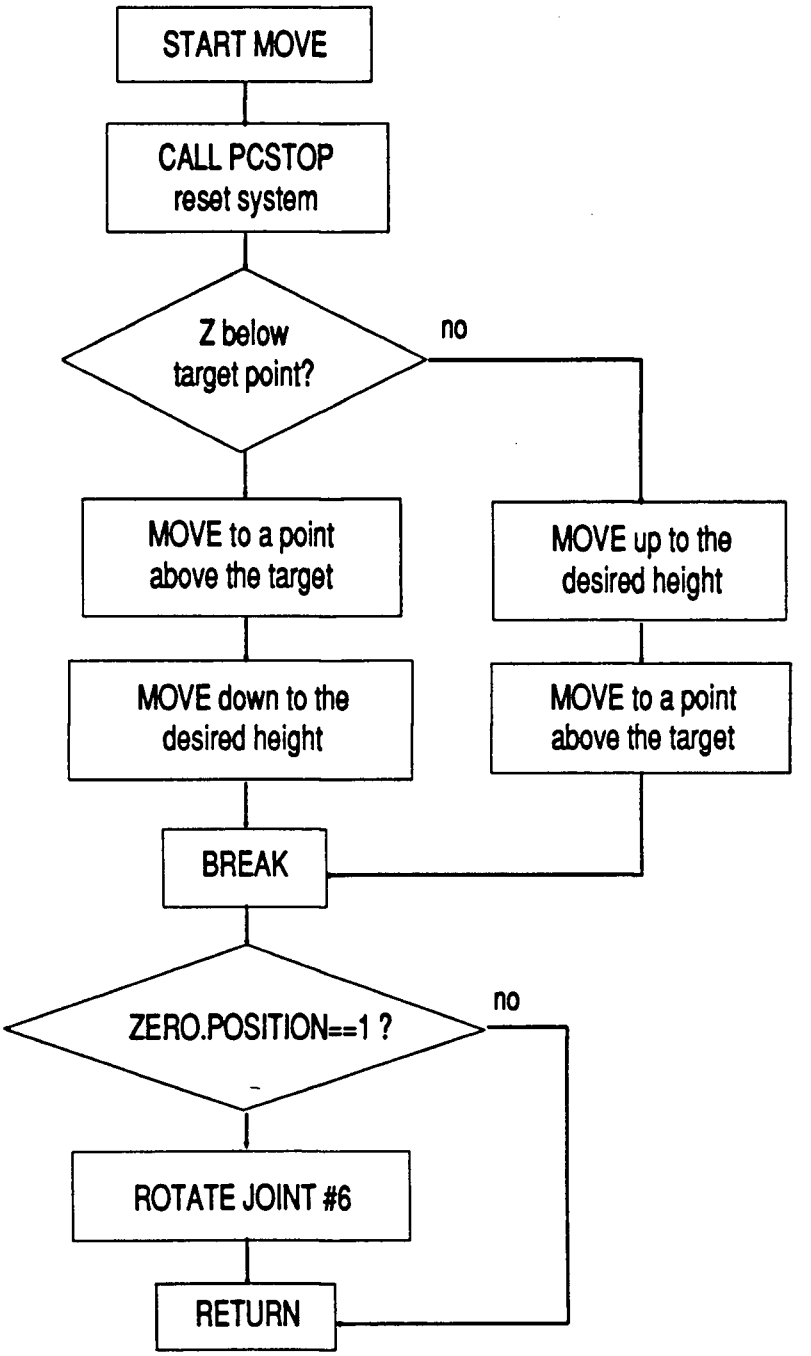

Figure 6.1: Subroutine Move flow chart.

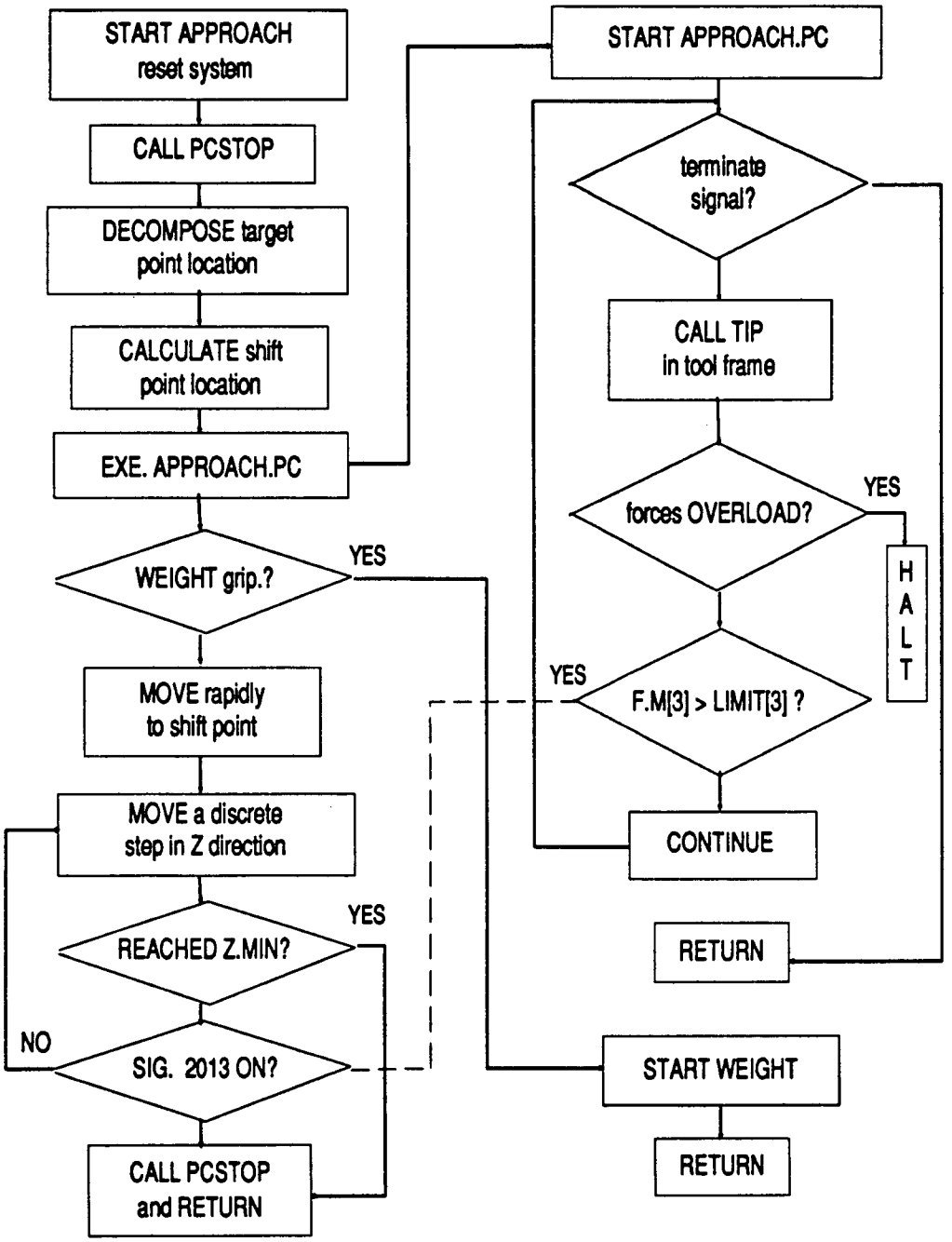

Figure 6.2: Subroutine Approach flow chart.

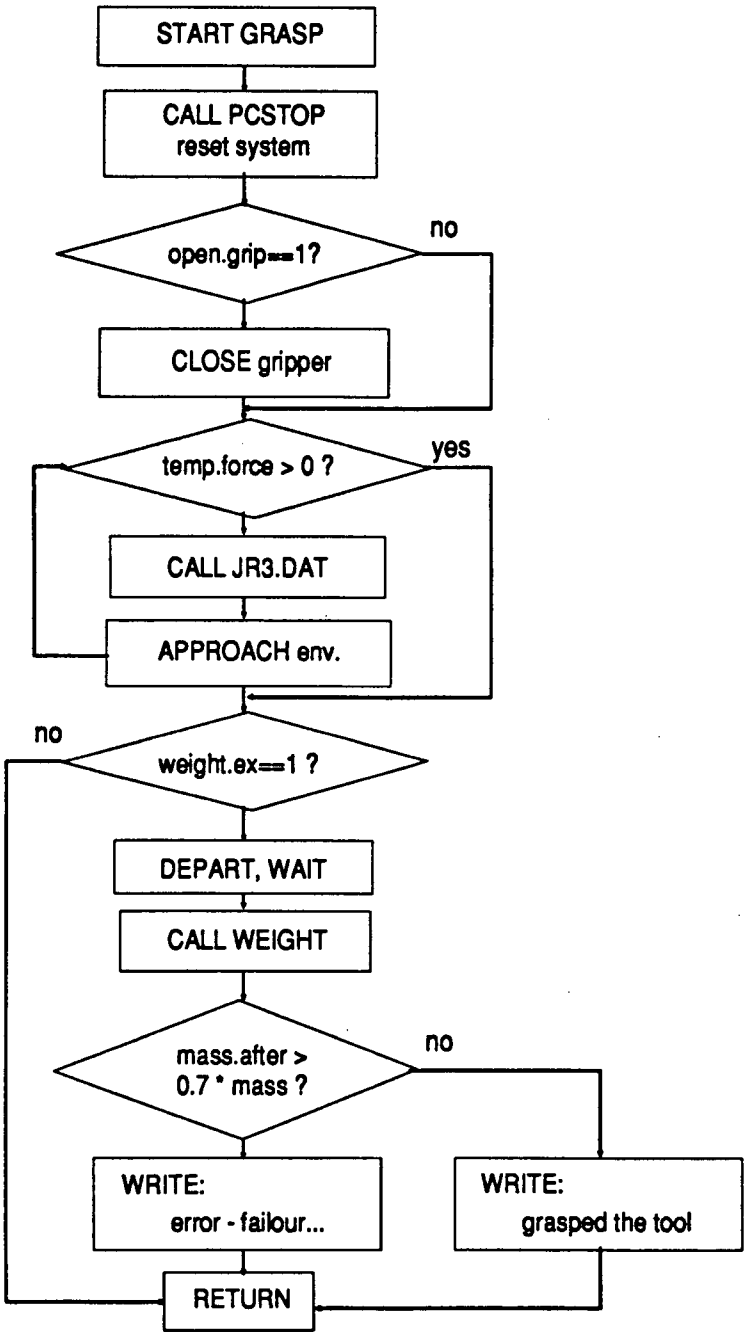

Figure 6.3: Subroutine **Grasp** flow chart.

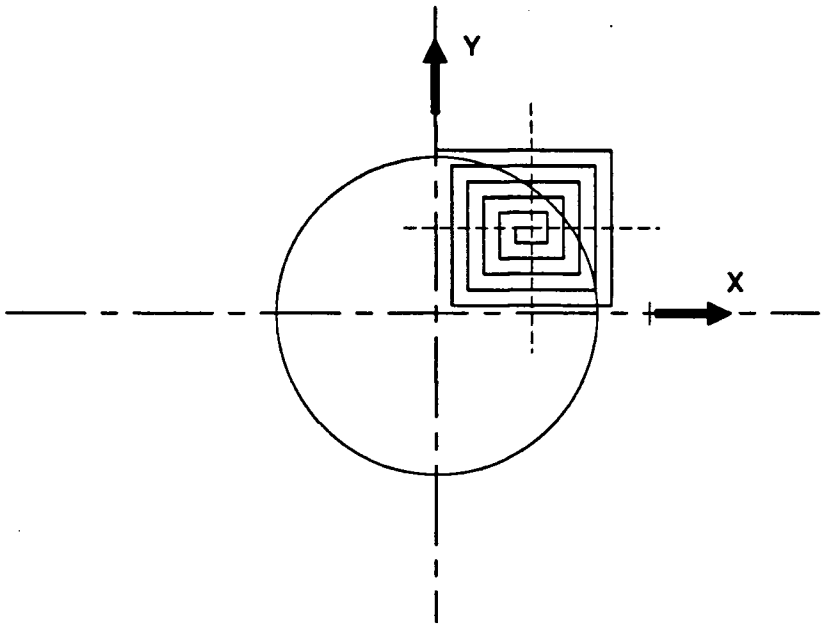

Figure 6.4: Search pattern.

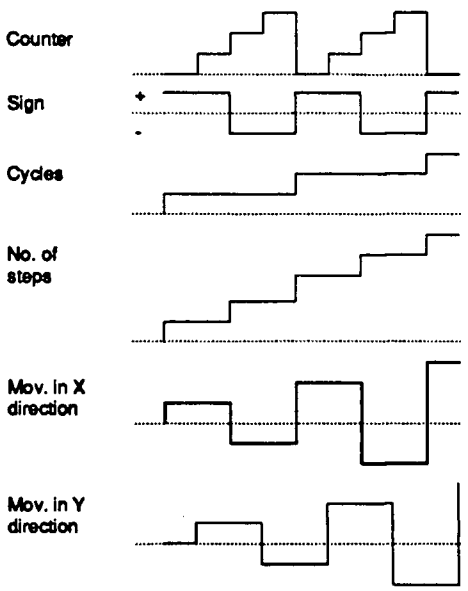

Figure 6.5: Search parameters.

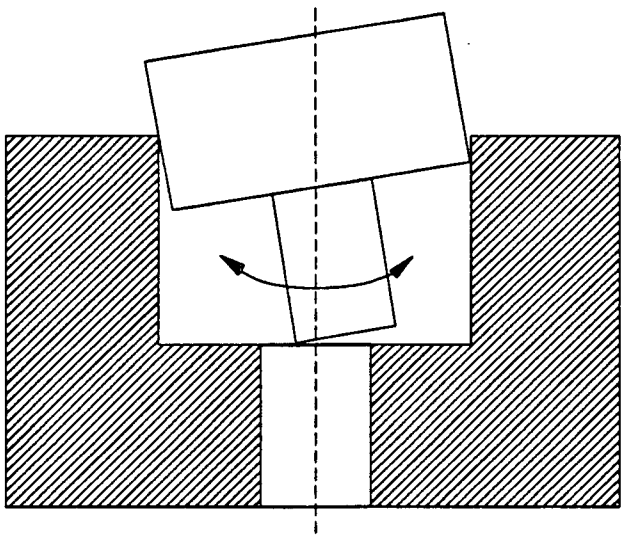

Figure 6.6: Successive insertion.

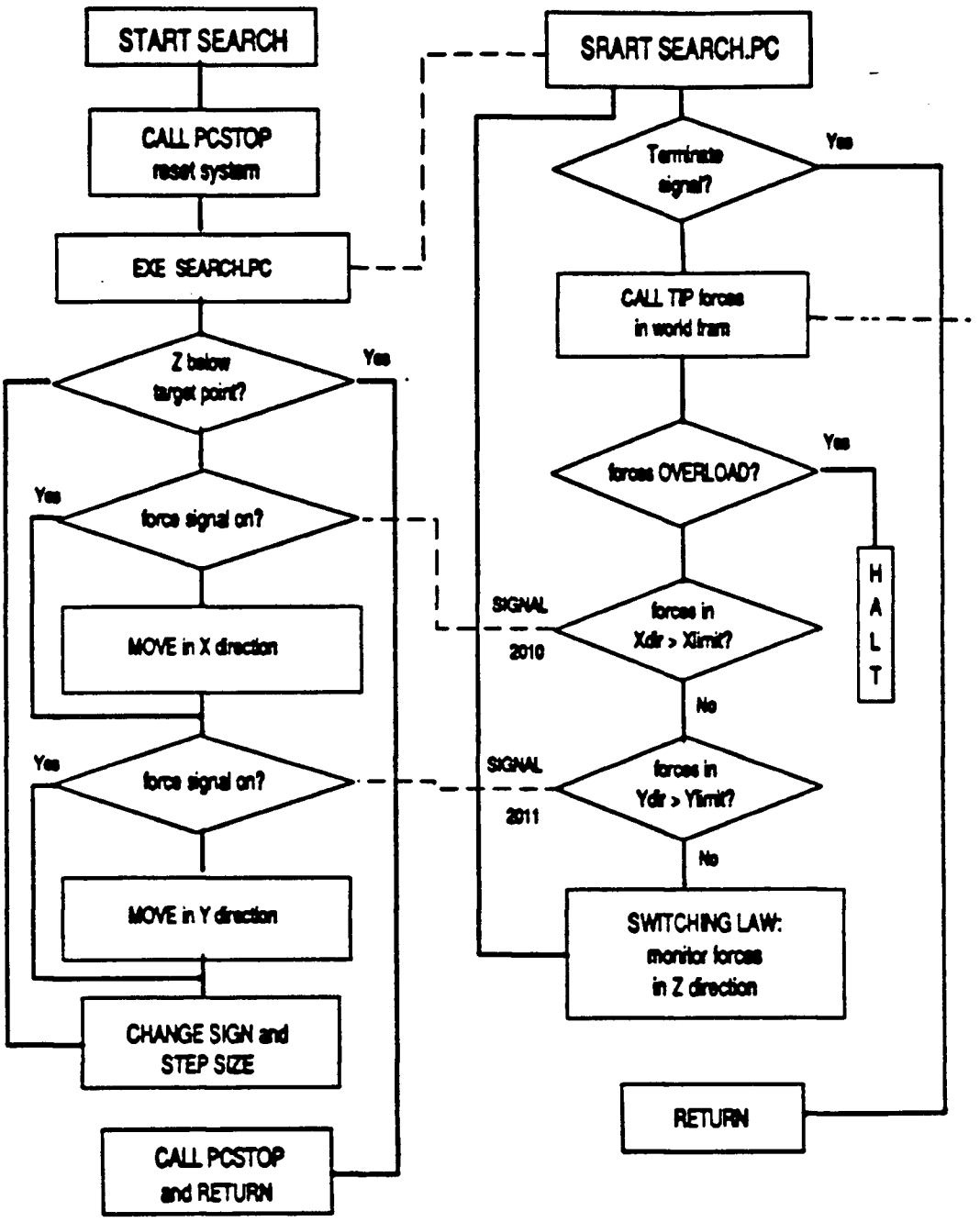

Figure 6.7: Subroutine Search flow chart.

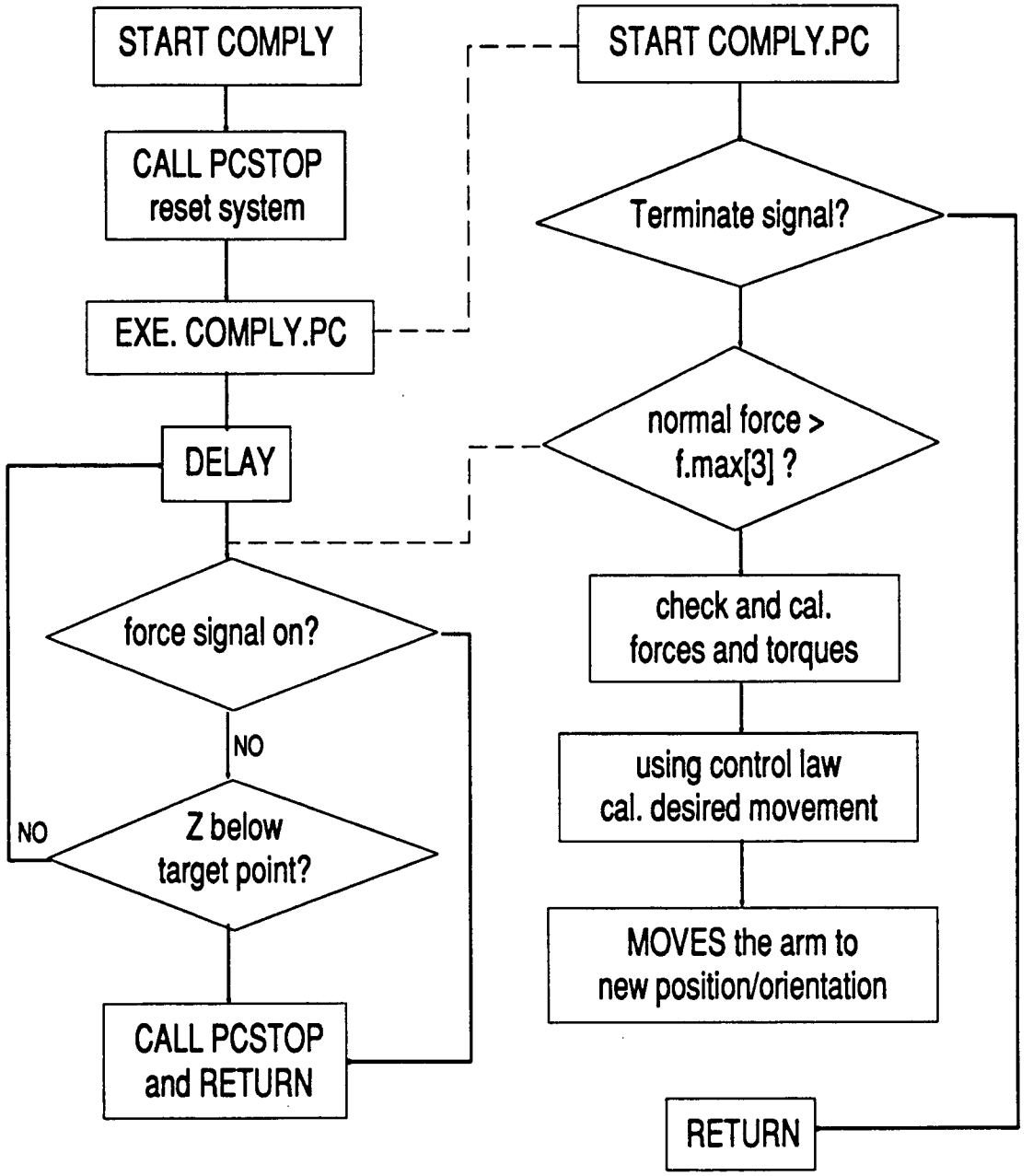

Figure 6.8: Subroutine Comply flow chart.

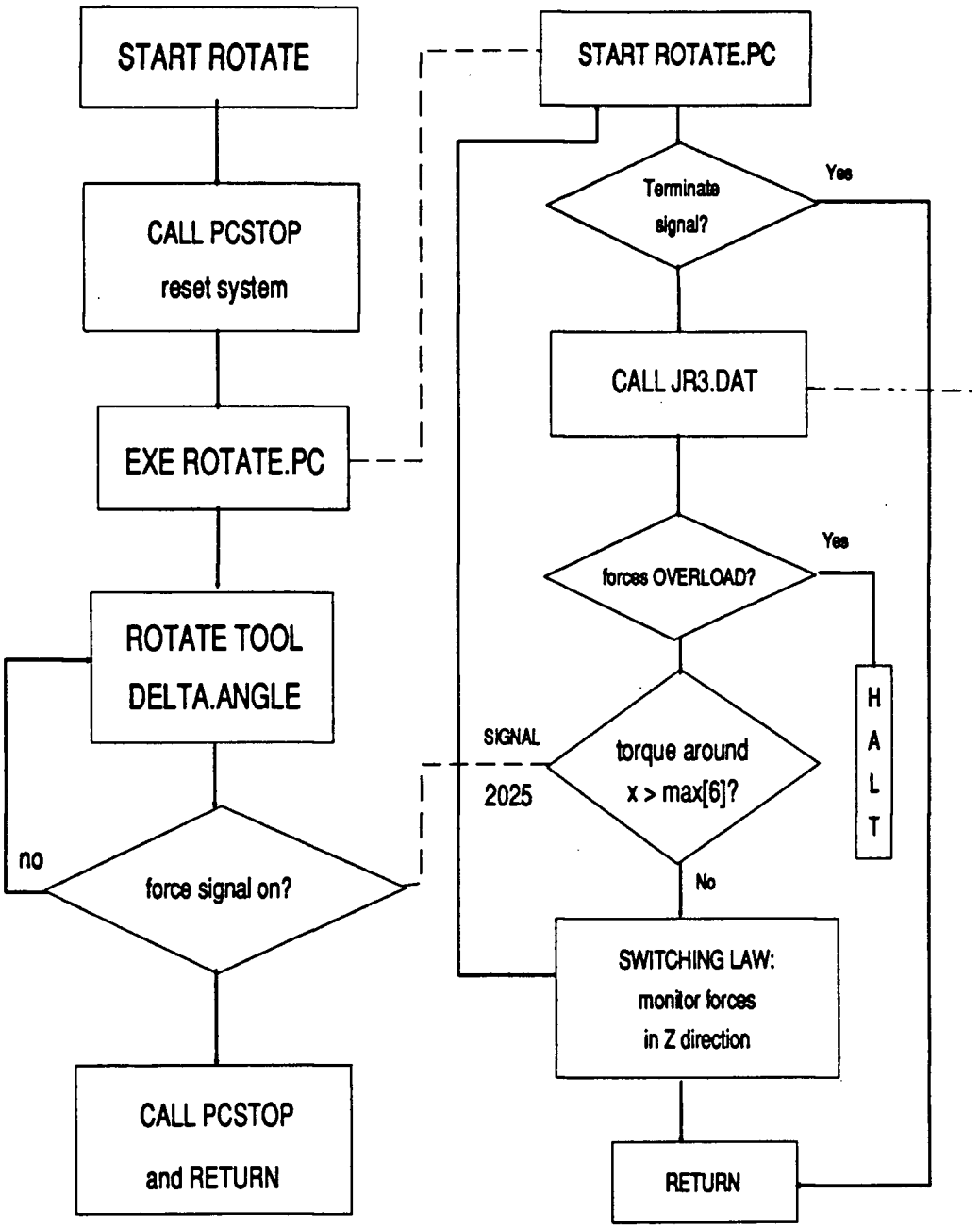

Figure 6.9: Subroutine Rotate flow chart.

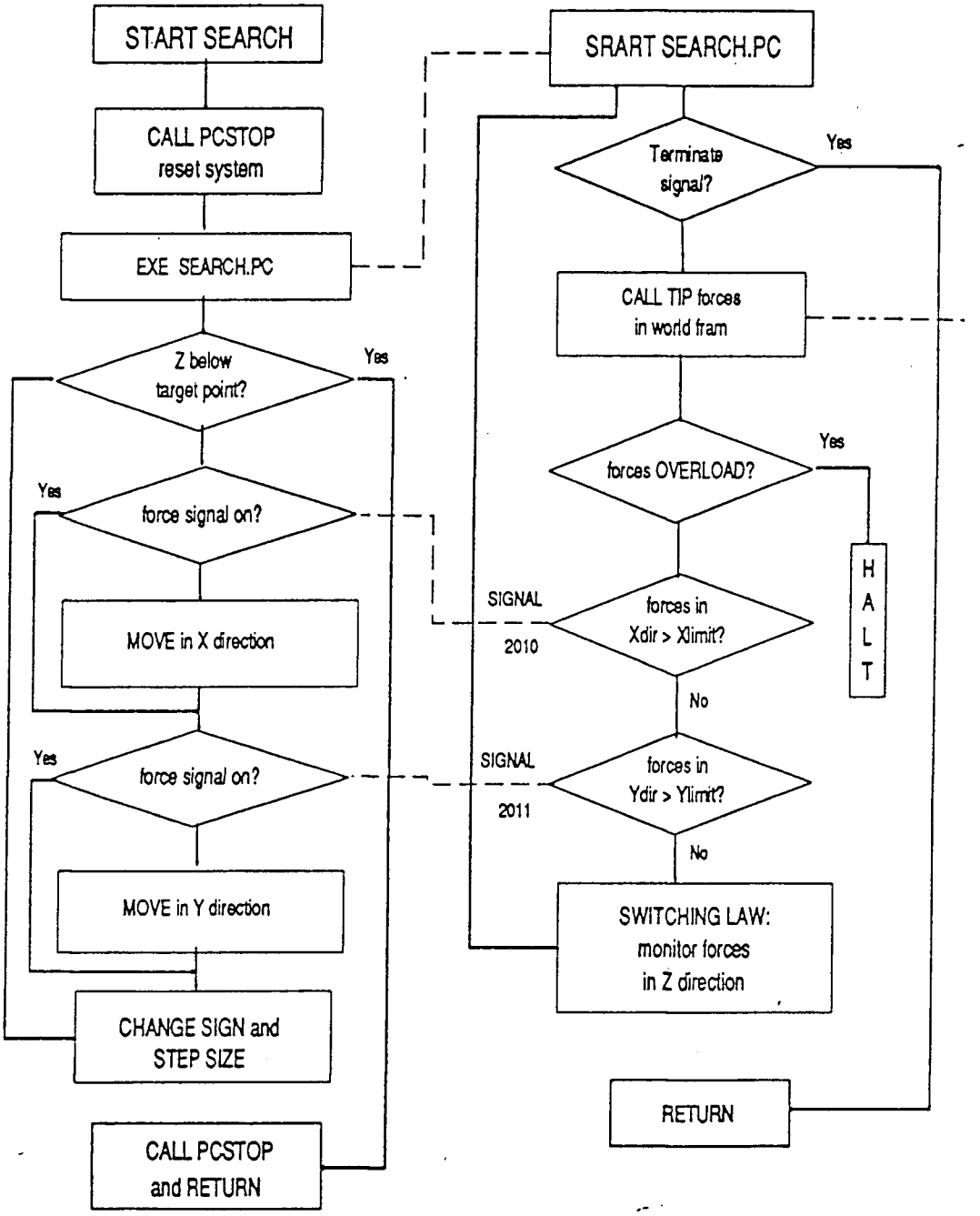

Figure 6.10: Subroutine **Clear** flow chart.

Chapter 7

SERVICE PROGRAMS

7.1 PROGRAM PCstop

PCstop program was created in order to reset parameters that are repeatedly used by the programs of the library. Executing the program helps to avoid problems like moving the arm while an undesired pc program is still active or receiving interrupts from unexpected signals sources. While examining the robot, it is good practice to call PCstop at the beginning and at the end of every subroutine. While running a debugged task, the call instruction can be used only at the beginning. PCstop includes the following functions:

1. NOALTER - terminate real-time control of robot motion (the pc program may still running but without an effect on arm movement).

2. PCEND - signals the process control program to stop at the end of its current execution cycle.

3. SIGNALS - turns some specific internal software signals on and off. All those signals are used for communications between subroutines and their related pc programs.

7.2 PROGRAM Null.sensor

There are some tasks where force information is required as a relative to the previous data and in other task a absolute force information is required. Using Grasp subroutine for example, in order to know whether the part was grasped by the arm, the end effector

weight can be measured before and after the grasp action. In this case, it is quicker to null the output of the sensor before the grasp, and to compare the absolute measured weight with the actual weight, stored in the initialization file. Another important use of the null function lies in the fact that there is a drift in the force information during use. Although there is an automatic gain drift compensation, software controlled through the serial port, this option can not be realized in real time, using VAL-II instructions.

To reset all force signals, the program sends an external signal, that activates the first relay of the I/O Module. The relay (using the digital ground - pin 24) transfers high bit to the Reset offsets pin (pin no. 5) of the discrete I/O connector located on the JR3 electronic support system. This action executes a soft reset routine that clears error and trips point flags and loads envelope latched status byte and by this, reset force vector.

7.3 OVERLOAD CHECKING

Both the robot controller and the force sensor data translator (**JR3.DAT**) have built in functions that react whenever overload occurs. The robot controller reacts by removing the electric power of the arm. The force sensor reacts by sending bad data envelope or, for less severe cases, by printing an error massage on the monitor screen.

For some delicate tasks, maximum forces have to be carefully controlled. Those forces are much lower than those of the system and for practical reasons, built in overload protections are not to be changed. Upon request, the program **Overload** simply check the received force information and compare it with the another list of maximum forces. If a bigger force is detected, a signal is transferred to the calling program and an immediate termination can be executed.

7.4 FORCE INFORMATION

Applying a synchronous, real time, on line trajectory changes (path control) to the PUMA 560 using VAL-II, can only be done by using *Process Control program.* This feature of VAL-II is extensively used while task specifications are a function of the data generated by a sensor. In our case, pc programs are used to modify the robot desired locations (while it is moving), using force data information. In this mode, the main program is responsible to move the arm according to the desired task positions and to activate or terminate the pc program. The pc program is responsible for receiving force information, comparing it with the desired forces and subjecting the force errors to the desired control law (as above). The set of new desired locations, that represent force errors in the force controlled axes, is used to alter arm movement (in all axes) that is currently controlled by the main program (using position control).

A pc program is very powerful primarily because it uses real time path control, executes in parallel and synchronized with the main program. ALTOUT is the VAL-II instruction that generates path modification (while the robot is moving) by sending 6 parameters corresponding to the X, Y and Z displacement data and X_r, Y_r and Z_r rotational data. The input data words (2 bits) specify the amount by which VAL-II modifies the nominal tool-tip trajectory using scale factors. The scale factors yield a 16 bit words and their values are 32 for distances (using [mm]) and 182 for rotations (using [deg.]). For the translation components the trajectory is modified by adding the ALTOUT values to the nominal robot location. For large rotations in noncumulative mode, the change in tool tip orientation is computed by first rotating about the X axis, then the Y axis and finally the Z axis by the specified amounts.

Force information is translated to position instructions using the control law as described previously. This section describes the way force data, received in the wrist sensor,

is manipulated and transformed to the desired coordinate frame and location. Two coordinate frames are being used in the library to control arm movement:

1. *TOOL coordinate frame* - is used for applications where the trajectory is altered by the process control program and no location data is needed for other programs. Complying the arm to the orientation of the hole during inserting task is a good example because this information used by the pc program and no other users needs it.

2. *WORLD coordinate frame* - is used when force information has effect on the following programs and is easy to understand, calculate and manipulate the arm according to this information. Searching program or Edge Follower are programs that are easy to control using world coordinate frame.

Force/torque information, received by the sensor, is the sensor's reactions to forces and torques mostly acting on the tip of the tool of the part being held in the gripper. This information is amplified, filtered, digitized, processed and sent via both serial and parallel communication lines. The robot, on his communication side, has to receive, check and process that data. JR3 company supplied with the force sensor an interface program called GETDATA. The program provides the necessary VAL-II code to retrieve forces and torques data from JR3 force sensor system's DMA communication link. The program was changed to a new program (JR3.DAT) that has the following features:

- checking received data vector for control words (start word and trailer word) and checking words (checksum and maskword). This is a check that the data received by the robot is the same as the data that was sent by the sensor. The checking part includes repeat operations and messages to the operator.

- checking overload for the force sensor (or other parts - using the desired transformations).

- using the dimensions of the part held by the gripper (tool, peg, screw etc.) the program uses the Jacobian to transform force information from the sensor's coordinate frame to the tool's coordinate frame. The data is stored in a force/torque array (f.m[1]...f.m[6]). This part is described in the following subsections).

7.4.1 Force/Torque data in tool coordinate frame:

The relative position between the force sensor location and an arbitrary point of interest is shown in Fig. 7.1. Typically, the point will be the contact point between the tool and the environment. The relative position can either be measured (assuming rigid body construction between the gripper and the object) or it can be calculated using forces and stiffness data.

Assuming two sets of generalized, independent coordinate frames $\mathbf{q} = [q_1, ..., q_n]^T$ and $\mathbf{p} = [p_1, ..., p_m]^T$ and their corresponding forces and torques $\mathbf{Q} = [Q_1, ..., Q_n]^T$ and $\mathbf{P} = [P_1, ..., P_m]^T$. In order to transfer forces and torques from one coordinate system to the other, a virtual displacement $\lambda\mathbf{p}$ is considered to take place at the contact point. Considering a complete set of generalized coordinates $\lambda\mathbf{q}$ at the sensor, the mapping is function is a $m \times n$ Jacobian matrix related with the coordinate transformation:

$$\lambda\mathbf{p} = \mathbf{J}\lambda\mathbf{q} \tag{7.1}$$

Infinitesimal translations and rotations of the rigid body are represented by a six dimensional vector:

$d\mathbf{q} = [dx, dy, dz, d\phi_x, d\phi_y, d\phi_z]^T$ with respect to the tip's coordinate frame $O - uvw$ and by the vector:

$d\mathbf{p} = [du, dv, dw, d\phi_u, d\phi_v, d\phi_w]^T$ with respect to the sensor coordinate frame $O' - xyz$. In matrix form, the transformation from $d\mathbf{q}$ to $d\mathbf{p}$ is given by

$$\begin{bmatrix} du \\ dv \\ dw \\ d\phi_u \\ d\phi_v \\ d\phi_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & r_z & -r_y \\ 0 & 1 & 0 & -r_z & 0 & r_x \\ 0 & 0 & 1 & r_y & -r_x & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \\ d\phi_x \\ d\phi_y \\ d\phi_z \end{bmatrix} \tag{7.2}$$

Force/torque processed data vector is received from the sensor by a program called **GET.DATA**, supplied by JR^3 company. In order to receive a force/torque vector at the contact point (usually at the tool's tip), the program was modified. The generalized forces $\mathbf{P}$, measured by the sensor, are transformed to the generalized forces $\mathbf{Q}$ at the tool's tip by

$$\mathbf{Q} = \mathbf{J}^T\mathbf{P} \tag{7.3}$$

or, in matrix notation

$$\begin{bmatrix} f.m[1] \\ f.m[2] \\ f.m[3] \\ f.m[4] \\ f.m[5] \\ f.m[6] \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -r_z & r_y & 1 & 0 & 0 \\ r_z & 0 & -r_x & 0 & 1 & 0 \\ -r_y & r_x & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} F_u \\ F_v \\ F_w \\ M_u \\ M_v \\ M_w \end{bmatrix} \tag{7.4}$$

JR3.DAT is a modified program. The geometric specifications of the tool: $\mathbf{r} = (r_x, r_y, r_z)^T$ must be added in order to receive the transformed force/torque vector.

The program **SHOW.FM** enables to see on the monitor force information received

from **JR3.DAT** (or any other force transformation program). This program was created mostly for checking and calibrating the force sensor and for checking force data transformations.

7.4.2 Force data using world coordinate frame:

For some applications, it is convenient to control arm movement using world coordinate frame (Fig. 7.2). Mostly, the relevant force information is located at the tip of the tool. So, force data must be transformed from the wrist sensor to the contact point, and there to be transformed to world coordinate frame.

The end effector location (or the tool's location - if its parameters are specified) is expressed in VAL-II using the following parameters: **X,Y,Z** positions in right hand world coordinate frame and 3 Euler angles **O**rientation, **A**ltitude and **T**ool (**O.A.T.**). This set of angles (used by UNIMATION) corresponds to the following sequence of rotation (refer to Fig. 7.3):

1. a rotation of O angle about the OZ axis ($R_{Z,O}$) .
2. a rotation of A angle about the rotated OV axis ($R_{V,A}$) .
3. a rotation of T angle about the rotated OW axis ($R_{W,T}$) .

The initial alignment of tool coordinate frame **n,s,a** relative to XYZ is: the hand points (a axis) to the negative y_0 axis and s axis points to the positive x_0 axis. The transformation that describes the orientation of n,s,a with respect to base coordinate frame x_0, y_0, z_0 is received by rotating around z_0 axis and then around y_0 axis:

$$R_{z_0,\theta=-90}R_{y_0,\phi=90} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix} \tag{7.5}$$

Matrix **T** is a 4×4 homogeneous transformation matrix that maps a vector expressed in *OUVW* coordinates system to *OXYZ* coordinate system

$$\mathbf{P}_{XYZ} = \mathbf{T}\mathbf{P}_{UVW} \tag{7.6}$$

using tool notation

$$\mathbf{T} = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} & & & p_x \\ & \mathbf{R} & & p_y \\ & & & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.7}$$

the rotational part **R** can be defined using the Euler angles that are resolved from the PUMA controller:

$$\mathbf{R} = \begin{bmatrix} n_x & s_x & a_x \\ n_y & s_y & a_y \\ n_z & s_z & a_z \end{bmatrix} = R_{z,O} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix} R_{s,A} R_{a,T} =$$

$$\begin{bmatrix} cO & -sO & 0 \\ sO & cO & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} cA & 0 & sA \\ 0 & 1 & 0 \\ -sA & 0 & cA \end{bmatrix} \begin{bmatrix} cT & -sT & 0 \\ sT & cT & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} -sOsAcT + cOsT & sOsAsT + cOcT & sOcA \\ cOsAcT + sOsT & -cOsAsT + sOcT & -cOcA \\ -cAcT & cAsT & -sA \end{bmatrix} \tag{7.8}$$

Using **JR3.DAT** transfers both forces and torques from force sensor location to the desired contact point. In order to change this data into world coordinate frame, the program **WORLD.DAT** has to use just the rotating part (**R**) of the homogeneous matrix **T**. The following matrix describe the rotation transformation of the e-e (or tool's

tip) of the Puma into world coordinate frame:

$$\begin{Bmatrix} du \\ dv \\ dw \\ d\phi_u \\ d\phi_v \\ d\phi_w \end{Bmatrix} = \begin{bmatrix} n_x & s_x & a_x & 0 & 0 & 0 \\ n_y & s_y & a_y & 0 & 0 & 0 \\ n_z & s_z & a_z & 0 & 0 & 0 \\ 0 & 0 & 0 & n_x & s_x & a_x \\ 0 & 0 & 0 & n_y & s_y & a_y \\ 0 & 0 & 0 & n_z & s_z & a_z \end{bmatrix} \begin{Bmatrix} dx \\ dy \\ dz \\ d\phi_x \\ d\phi_y \\ d\phi_z \end{Bmatrix} \tag{7.9}$$

In the first part of WORLD.DAT, the vector $f.m[1...6]$ (received from JR3.DAT) is labeled as a new vector ($temp[1...6]$). After rotational transformation, the name of the output vector has the same name as the output vector of JR3.DAT. In this way, the following programs can be operated with either programs without any changes.

Once again, the transposed Jacobian is applied to the translated forces and torques at the contact point, in order to resolve it into world coordinate frame:

$$\begin{Bmatrix} f.m[1] \\ f.m[2] \\ f.m[3] \\ f.m[4] \\ f.m[5] \\ f.m[6] \end{Bmatrix} = \begin{bmatrix} n_x & n_y & n_z & 0 & 0 & 0 \\ s_x & s_y & s_z & 0 & 0 & 0 \\ a_x & a_y & a_z & 0 & 0 & 0 \\ 0 & 0 & 0 & n_x & n_y & n_z \\ 0 & 0 & 0 & s_x & s_y & s_z \\ 0 & 0 & 0 & a_x & a_y & a_z \end{bmatrix} \begin{Bmatrix} temp[1] \\ temp[2] \\ temp[3] \\ temp[4] \\ temp[5] \\ temp[6] \end{Bmatrix} \tag{7.10}$$

the notation parameters ($n_x...a_z$) are defined by equation 7.8.

7.5 PROCESS CONTROL PROGRAMS

In the previous chapter, subroutines were described. The relations between subroutines and pc program in controlling arm movement lays in the fact that pc programs modify

arm path that was assigned by subroutines. This real time path modification is achieved by applying force data to an appropriate control algorithm. Every pc program contains call instructions that call for force data in tool or world coordinate frame. The position control movements of the arm are altered by real time path modifications that are calculated using proportional control law and appropriate parameters (received from the initialization file).

In chapter 4 the basics of the control law were discussed. In this section, a discrete time analysis is performed, in order to get the best relations between the stiffness of the parts taking part in the assembly process (arm structure, sensor, tool and environment - $\mathbf{K}_s$) and the software gains. Referring to Fig. 4.3, the contact forces across the sensor ($\mathbf{F}_s$) are the output variable of the closed loop system ($\mathbf{F}_a$)

$$\mathbf{F}_a = \mathbf{K}_s \mathbf{X}_s \tag{7.11}$$

Implementing a proportional control law in the system, the feedback gain $\mathbf{K}_{fd}$ multiplies the difference between the desired force $\mathbf{F}_d$ and the actual force $\mathbf{F}_a$. The parameter $\mathbf{K}_{fd}$ serves as gain that changes force errors into position errors.

$$\mathbf{X}_e = \mathbf{K}_{fd}(\mathbf{F}_d - \mathbf{F}_a) \tag{7.12}$$

Using a discrete time relation between the current and the previous positions

$$\mathbf{X}_{s(k+1)} = \mathbf{X}_{s(k)} + \mathbf{K}_{fd}(\mathbf{F}_d - \mathbf{F}_a) \tag{7.13}$$

and changing arm stiffness parameter into a serial of springs (indicate serial connections of some stiffness like arm, sensor, compliance device etc.)

$$\mathbf{X}_{s(k)} = \mathbf{F}_{a(k)}\left(\frac{1}{\mathbf{K}_1} + \frac{1}{\mathbf{K}_2}\right) \tag{7.14}$$

$$\mathbf{X}_{s(k+1)} = \mathbf{F}_{a(k+1)}(\frac{1}{\mathbf{K}_1} + \frac{1}{\mathbf{K}_2})$$

using equation 7.14 yields

$$\mathbf{F}_{a(k+1)}(\frac{1}{\mathbf{K}_s} + \frac{1}{\mathbf{K}_E}) = \mathbf{F}_k(\frac{1}{\mathbf{K}_s} + \frac{1}{\mathbf{K}_E}) + \mathbf{K}_{fd}\mathbf{F}_d - \mathbf{K}_{fd}\mathbf{F}_a \tag{7.15}$$

or

$$\mathbf{F}_{a(k+1)} = \mathbf{F}_{a(k)}(1 - \frac{\mathbf{K}_{fd}}{(\frac{1}{\mathbf{K}_1} + \frac{1}{\mathbf{K}_2})}) + \mathbf{F}_d(\frac{\mathbf{K}_{fd}}{(\frac{1}{\mathbf{K}_1} + \frac{1}{\mathbf{K}_2})}) \tag{7.16}$$

The following can be concluded:

1. between the wrist force sensor and the environment in contact, there is the mass of the sensor, end effector and a part (or a toll). Transient motions of those masses are received by the force sensor as reaction forces. The above discussion ignores this input and is implemented just to relatively slow, in contact tasks.

2. the net added force is a function of the gain and the serial stiffness. Smooth movements can be achieved by either low gain or by a compliance device. When a compliance device is not added to the system and there is no contact with the environment, the software gain ($\mathbf{K}_{fd}$) has to receive high values so that the system can response quickly (**Approach** Subroutine).

3. decreasing time interval between cycles (every cycle includes sampling, calculations and actual movement) can improve drastically the reactions of the system, using low software gains without a compliance device.

When the proportional control algorithm is used in a pc program like **Comply.pc**, it is applied to the six degrees of freedom at the same time. It means that there is coupling between the movements and the designer has to consider parameters like friction when calculating the input parameters of each control low. The parameters are: desired force

($\mathbf{F}_d$), proportional gain ($\mathbf{K}_{fd}$), dead zone around the desired force (DZ) and the bounded forces ($\mathbf{F}_{max}$,$\mathbf{F}_{min}$).

7.6 INITIALIZATION and PARAMETRIC FILES

Every subroutine and part of the service programs, use a list of parameters during their execution. In the library those parameters are stored using two categories: Parametric and Initialization files. Parametric files are program oriented and include all the parameters the program is using during its execution (actually it is part of the program that was changed into a file). Initialization files are task oriented and contain only those parameters that are to be change before a specific task is executed. As described earlier, the first part of task execution is to call to the Initialization files. Each file is divided according to the subroutines of that specific task. For each subroutine, the appropriate Parametric file is called, followed by a list of task oriented changes. Fig. 3.4 shows a block diagram of programs relations while Fig. 7.4 includes the beginning of an initialization file.

This structure is not the simplest but it is modular and relatively low memory consuming. It allows the designer to define all the basic and the desired parameters *before* the actual execution of every subtask. Building the library in this structure means that there are not stand alone programs. Using a subroutine without activating properly its parametric files can cause problems because the manipulator controller will use the last declarations that remains in its memory. It means risky movements to unexpected locations with unexpected speeds.

Every Initialization program begins with part configuration that serves as input for force manipulation (previously discussed) and Tool transformation. Every subtask has its own tool or part being held and activating by the arm. Mostly, While executing a program, the interested part of the arm is the tool's tip and it has to be declared in

a **TOOL** transformation. This relative tool transformation is automatically taken into consideration each time the location of the robot is requested, when a command is issued to move the robot to a location defined by a transformation. If no transformation value is specified, the tool transformation is set equal to "null tool". This transformation has its center at the surface of the mounting flange of the sixth joint. Tool transformation must take into consideration the sizes of the force sensor, RCC, gripper and any other devices attached to the last joint. If "teach" mode is used to record points locations to the robot memory, it must be done using the same Tool transformation as in actual execution. Once again, risky location may be reached using improper Tool transformation.

7.7 PROGRAM WEIGHT

A simple and quick way to check whether a **Grasp** task succeeded is by checking the weight of the part hanged on the force sensor and to compare it with the data received before the Grasp action. If the program **Null.sensor** was used before the **Grasp** action, the net weight of the grasped part is received from the program. The weighted data is then compared with the actual weight of the part (must be located in the active initialization file). If the result is lays between the boundaries, specified by the program - **Grasp** operation is confirmed and control may advanced to the next step. If the force information is lower then the expect value - another **Grasp** attempt is activated. Two unsuccessful grasps attempts transfer control either to **Search** pattern or terminate task execution.

One of the problems with weighting the part was arm acceleration or deceleration and its influence on the accuracy of the results. it is very important to avoid a next try to grasp the part because the it is already held by the end effector and a new approach will cause hand opening and possible damage to the part and environment when it fall down.

To avoid arm movements during the actual force measurements (**Jr3.dat**), a BREAK and WAIT (1 [sec]) instructions are executed.

In order to receive accurate data, the force sensor is sampled 25 times (can be changed in the initialization file). The information is averaged and only those inputs that their values are in the region of $\pm 20\%$ are used for a new set of averaging. The waiting and averaging make the program time consuming, and it is a good examples how the use of a vision system, instead of force sensor, can drastically save time and increase reliability.

7.8 PROGRAM REPEAT

For repeating operations like inserting the four screws in their holes in the top of the pump, a repeat procedure can save memory space and ease programing and debugging. **Repeat**, like an initialization file, is a special purpose program that is created for special task and has certain, fixed structure. The program has an extension according to the task that is repeated and in this way several programs can be located in the same subtask.

The program will be located after the initialization tasks and the subroutines that are used to perform the repeated task, so this program will execute the second and the following repetitions. The program uses a do-loop to call the initialization programs, after updating the desired information the desired subroutine is called until the task is completed.

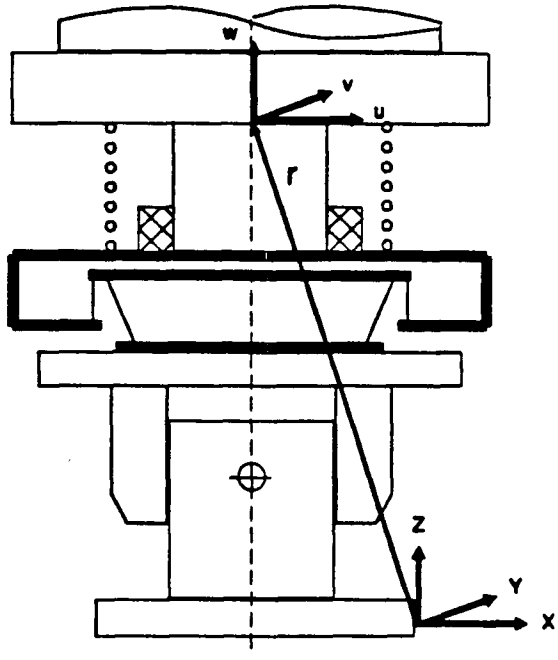

Figure 7.1: Rigid body transformation.

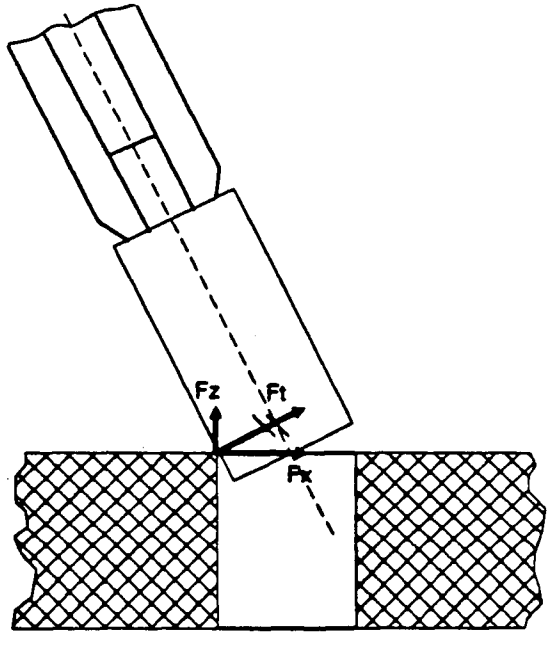

Figure 7.2: Force/torque rotation into world coordinate frame.

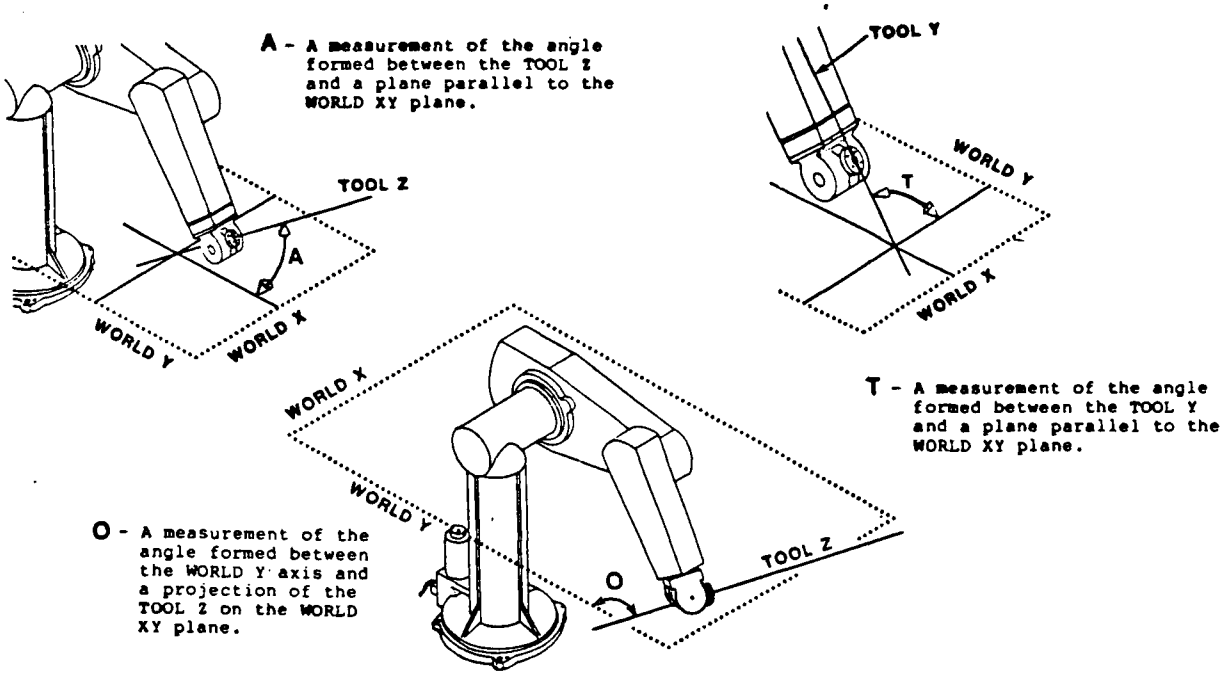

Figure 7.3: Orientation, Altitude and Tool (O.A.T.) angles.

```
1 ; GEAR1.INI2 (May. 24 /Kotzev)
2 ;
3 ; TOOL configuration:
4   z.sensor = 31
5   z.rcc = 43
6   z.gripper = 112
7   TOOL TRANS(0,0,z.sensor+z.rcc+z.gripper,90,-90,45)
8   mass=0.18
9   ds[1] = 8
10  ds[2] = 8
11  z.tool = 57
12  d[3] = z.rcc+z.gripper+z.tool
13 ; MOVE parameters:
14  CALL move.par
15  SET tool.pnt = gear1.pnt2
16  safe.z = 300
17 ; APPROACH parameters:
18  CALL approach.par
19  weight.ex = 1
.   .   .
.   .   .
.   .   .
```

Figure 7.4: An Initialization file structure example.

Chapter 8

FORCE CONTROL SIMULATION

Most of the algorithms that solved the dynamics of robotic arms are derived for an open kinematic chain configuration. In a case that refers to any manipulation task in which the gripper is not allowed to move freely but its motion is subject to some constraints resulting from the interaction with the environment - a closed kinematics (and dynamics) chain is to be solved. Ref. [41] and [42] consider that problem including analysis of manipulator with constraints on gripper motion.

In order to predict the dynamics of simple force-controlled robot system and to understand it's stability, the authors of Ref. [11] developed a series of lumped-parameter models that included effects of robot structural dynamics, sensor compliance and workpiece dynamics. Checking some models that included combinations of the robot dynamics and the dynamics of its workpiece, an instability has been shown to exist for robot models which include representation of a first resonant mode of the arm. The mode modeled could be attributed to either drive train or structural compliance (or both). The instability is present because the sensor is located at the point remote from the actuator. The controller then attempts to regulate contact force through a dynamic system.

In order to have a better understanding in the force and position control of the arm, a numerical simulation was coded. The model simulates the end effector, the discrete controller of the robot, the control law, the sensor and the environment. A guarded move task was chosen, because it is used frequently in assembly tasks and is a classic case to apply both position and force control algorithms to a robot manipulator. In the

simulation, parameters like arm compliance, sensor stiffness, environment position etc. can be changed and it's influence on task performance can be calculated. Running a subtask like plane searching (for hole, edge etc.) causes the tool to move in a square pattern, perpendicular to the plane, while increasing the rib's dimensions. The force sensor is used to sense normal and tangential force acting on the tool's tip. Under position control, the tool is moved in its $X - Y$ plane to a desired location and under force control the tool remains in contact (desired force) with the environment. Increasing in the plane (tangential) forces indicates an edge, an obstacle or a simple increasing of the friction forces that cause strategy changes according to the program's logic.

8.1 SYSTEM DESCRIPTION

8.1.1 Applied Force Control:

Using the PUMA 560 with its original controller and VAL-II language, a force control algorithm can be implemented through position commands. There are primarily two ways to receive and apply force information to the arm movement:

Robot control program: according to task specifications, the designer can build two control algorithms in the main program. Some axes (designate in world or tool coordinate frame) are controlled via force data, while the others are position controlled. While running the program, force information is received by calling **JR3.DAT** and then (if needed) the program calls **TIP** to transform force/torque information to the desired location. In the axes that are controlled via force information, the force data is compared to the desired forces. Subjecting the force errors to the task's control law, force errors are represented by new desired positions that can be understood and executed by the robot control box. After this stage all the axes (both position and force control), has motion representation. The robot controller can derive an inverse kinematic solution and move

the arm to the new desired location.

Process Control program: as described in the previous chapter, a pc program is very powerful primarily because it is a real time path control, parallel and synchronized with the main program. ALTOUT is the VAL-II instruction that generates path modification by sending 6 parameters corresponding to the X, Y and Z displacement data and X_r, Y_r and Z_r rotational data. The input data words (2 bits) specify the amount by which VAL-II modifies the nominal tool-tip trajectory using scale factors. The scale factors yield 16 bit words and their values are 32 for distances (using [mm]) and 182 for rotations (using [deg.]). For the translation components the trajectory is modified by adding the ALTOUT values to the nominal robot location. For large rotations in noncumulative mode, the change in tool tip orientation is computed by first rotating about the X axis, then the Y axis and finally the Z axis by the specified amounts.

Timing Considerations: according to Ref. [38], using internal alter (the altering mode of pc program), VAL-II expects an ALTOUT program instruction to be executed every 28 [msec] to pass control data to the robot motion controller. Because of computation time required by VAL-II to perform the transitions between motion segments, there is a limit on how closely spaced commanded locations can be. When locations are too close together, there is not enough time for VAL-II to compute and perform the transition from one motion to the next, causing a break in the continues path motion. This means that the robot stops instantaneously at intermediate locations. To prevent this, straight line motions can be used if the motion segment take more than about 140 [msec] or about 60 [msec] for joint interpolated motions.

8.1.2 Arm Stiffness:

As described previously, applying force control algorithm to the robot is done by introducing small movements to the end-effector while in contact with the environment.

When the desired set point is located a small segment into the surface, the controller will try to move the arm to that point. The joint motors will cause the arm to bend until the position error will null (joint encoders reaching the desired angles). The result of such pseudo movement will increase force/torque reactions. Decreasing force/torque reactions is done by moving the end-effector to the opposite direction.

When the use of the robot includes fast movements, structural dynamic characteristics of the manipulator including stiffness, inertia, damping and natural frequencies are to be considered. Using the arm while moving in contact with the environment involves fine movements so part of the dynamic characteristics can be ignored. In this case, force/torque reactions between the arm and the environment are a function of the step size and the stiffness of the contacting parts. Stiffness parameters that have to be taken into consideration are arm stiffness ($\mathbf{K}_{arm}$), stiffness combination of the sensor, gripper and the tool (held by the gripper) ($\mathbf{K}_{sen}$) and the stiffness of the environment ($\mathbf{K}_{env}$).

In order to be able to simulate an assembly task, parts stiffness parameters have to be calculated or measured. When working against a stable environment (the usual case while performing an assembly task), the stiffness values of both the sensor and gripper/tool combination can be measured or calculated, because they are linear and contact orientation is known. The robot arm, on the other hand, has many optional contact positions and orientations, so the nonlinear stiffness values are not constant. This problem can be solved in the following ways:

- stiffness envelope: for a particular robot a measurement stiffness envelope can be derived and applied according to the end effector locations and force/torque direction (assuming it is working at a known point). Using interpolation between known point can give good estimation to arm stiffness. A measurement of some principal points was done on the Puma in the CAMROL Lab. and is shown in Fig.

8.1.

- stiffness calculations: using reduced parameters equations (described in the following subsection) and approximated linear terms, arm Stiffness can be derived according to its position and orientation.

The assembled parts of the pump are very stiff, it cause the force control algorithm to move the arm using small steps. Dealing with tracking high sloped environment contours, small steps movement is relatively slow. One of the possible ways to improve system performance, is by introducing a compliance device (like linear spring) to the system. It has to be compliant relatively to the other components of the system, so their stiffness can be ignored. The compliant device can be located in places like the basis of the assembled structure or to the base of the robot. The simplest and easiest to install location is the manipulator wrist (shown schematically in Fig. 2.5).

8.1.3 Reduced Parameters:

A complex mechanical system like robot arm has many compliant components, inertia and energy dissipating units (dampers). Reflecting the values of all those components to a selected point (or part), enables to reduce the complexity of the system without affecting neither natural frequencies nor modes of vibration. The overall stiffness (or compliance) of a reduced model would be the same value received by the ratio of the applied force (or torque) to the resulting deflection (ref. [31]). Fig. 8.2 shows the arm (maximum outstretched) in rotation around a horizontal axis. The arm components compliance can be reduced to torsional compliance e_ϕ around the shoulder axis

$$e_\phi = \frac{\Phi}{Fl} = \frac{\Phi}{M} \tag{8.1}$$

where

e_ϕ - angular compliance ($\frac{1}{K_\phi}$).

Φ - angular deflection of the beam.

l - distance between load operation point to pivot.

using linear parameters, the linear compliance e_y is a function of the linear displacement y_f caused by the force F at the force application point (mostly the end effector):

$$e_y = \frac{y_f}{F} = \frac{\Phi l}{F} = \frac{e_\phi F l^2}{F} = e_\phi l^2 \tag{8.2}$$

The linear compliance can be transformed (and so it can be reduced) to any intermediate point at a distance a from the center

$$e_\phi = \frac{e_0}{a^2} = \frac{e_y}{l^2} \tag{8.3}$$

The overall reduced torsional compliance e_p of the robot in a considered mode, at a specific point p (like the end-effector) can be written

$$e_p = e_\phi l_p^2 = (\frac{e_0}{l_0^2} + \frac{e_1}{l_1^2} + ... + \frac{e_f}{l_f^2}) l_p^2 \tag{8.4}$$

where $e_0, e_1, ...$ are compliant parameters of the system and $l_0, l_1, ...$ are their distance from the pivot axis.

Calculation of a stretched PUMA 560 arm shows that its approximate compliance is about 10^{-5} [rad/Nm] or, using arm length of about 830 [mm], the approximate stiffness is 120 [N/mm]. Fig. 8.1 shows the data received from measurements of the arm in the lab. The measured data shows uniform force/displacement line. This stiffness is half the calculated stiffness, a reasonable result regarding the approximation in the calculation and the age of the robot (wear, backlash etc.). In the simulation the measured data is being used.

Using the same logic, mass reduction can be perform (Fig. 8.2) based on the expressions of the kinetic energy for the masses before and after reduction.

$$m_p = (m_0 l_0^2 + m_1 l_1^2 + ... + m_f l_f^2)\frac{1}{l_p^2} \tag{8.5}$$

where $m_0, m_1, ...$ are the initial masses, $l_1, l_2, ...$ are their distance from the pivot point and m_p is the reduced mass. Calculation of the reduced structural mass at the end effector must be based on estimations of joint masses and their locations (there is no information on that subject in Ref. [38]). Using those estimations and verifying the result by Ref. [31], 3.5 [Kg] was used in the simulation as the effective structural mass at the end effector.

Fig. 8.3 represents a linear model of the robot arm moving towards the environment. In this model both the arm structure, the sensor (plus a tool) and the environment are compliant and has some damping capabilities.

8.1.4 Impact:

An impact problem is defined to be the collision of two bodies with known energy loss and a contact force with defined direction. An impact between the tip of the tool and the environment is an essential part in Approach subroutine (part matting) and in the guarded move task. In this case we assume a central impact - the line of action of the contact forces passes through the mass center of each body so there is no rotation of either body. In the simulation, the movements of the contacting parts (sensor/tool and environment), were based considering the following:

a) Conservation of linear momentum (collinear case):

$$m_1(V_1 - v_1) = -m_2(V_2 - v_2) \tag{8.6}$$

where m_1, m_2 are the masses of the bodies and v_1, v_2, v_1', v_2' are their velocities before and after the collision.

b) Conservation of energy:

$$\frac{1}{2}m_1v_1^2 + \frac{1}{2}m_2v_2^2 = \frac{1}{2}m_1V_1^2 + \frac{1}{2}m_2v_2^2 + E \tag{8.7}$$

where E is the energy loss during impact (known). Using the coefficient of restitution $[e = (V_2 - V_1)/(v_1 - v_2)]$, the following equations give the velocity of each body following the collision:

$$\begin{aligned} V_1 &= \frac{m_1v_1 + m_2v_2 - em_2(v_1 - v_2)}{m_1 + m_2} \\ V_2 &= \frac{m_1v_1 + m_2v_2 + em_1(v_1 - v_2)}{m_1 + m_2} \end{aligned} \tag{8.8}$$

For a spacial case of collision between small mass and relatively big mass, (like the case of fixed environment), the change in the velocity of the big mass is negligible and the relative velocity becomes:

$$V_1 = -ev_1 \tag{8.9}$$

For steel (balls) the coefficient of restitution was found to be $e = 0.55$ and this value is used in the simulation.

8.2 MODEL DESCRIPTION

The force control algorithm applied in the simulation is according to the force control block diagram explained in chapter 4. The environment was derived so that hard tracking topography can be checked. In the simulation, movement in the X direction begins from the moment the arm begins with the approaching towards the environment. This gives a better orientation and timing while studying the results. Like in the Search subroutine, there is no coupling between the movement in the X and Z direction. i.e. using the arm in actual tracking a contour, if forces exceeded high values or normal forces becomes zero (no contact) - the actual movement is the sum of the force controlled movement in Z direction and the continuous, position controlled movement in X direction.

At time $T = t_0$, the arm begins a rapid movement towards a target point located 10 [mm] above the environment. The movement is achieved by using a discrete ($\Delta T = 0.028[sec]$) simulation and control low that add to the dynamic equations, the desired location and velocity. The time interval and the restricted desired steps of the arm prevent high forces due to high velocities and damping (without actual contact). After the first approach to the desired location, the vertical velocity is lowered to approx. 16 [mm/sec]. For fixed environment, the dynamic equations of the arm and the sensor (without the input from the control low) are:

$$\begin{bmatrix} m_r & 0 \\ 0 & m_s \end{bmatrix} \begin{Bmatrix} \ddot{z}_r \\ \ddot{z}_s \end{Bmatrix} = \begin{bmatrix} (b_r + b_s) & -b_s \\ -b_s & b_s \end{bmatrix} \begin{Bmatrix} \dot{z}_r \\ \dot{z}_s \end{Bmatrix} + \begin{bmatrix} (k_r + k_s) & -k_s \\ -k_s & k_s \end{bmatrix} \begin{Bmatrix} z_r \\ z_s \end{Bmatrix} \tag{8.10}$$

when

subscript r - arm parameters.

subscript s - sensor (or tip of the tool).

the equation for arm motion is:

$$m_r \ddot{z}_r = -\dot{z}_r b_r + (\dot{z}_s - \dot{z}_r) b_s - z_r k_r + (z_s - z_r) k_s \tag{8.11}$$

the control algorithm is added by adding the desired position (z_d) and desired velocity ($\dot{z}_d$) as input to the contact point (Fig. 7.3) of the spring and damper of the arm. The equation that simulate arm movements (in the simulation) is

$$m_r \ddot{z}_r = (\dot{z}_d - \dot{z}_r) b_r + (\dot{z}_s - \dot{z}_r) b_s + (z_d - z_r) k_r + (z_s - z_r) k_s \tag{8.12}$$

The second mass (sensor + tool) is moving according to the Eq. 8.10.

In the simulation, contact position is indicated by comparing the sensor's location to that of the environment. If at the end of a certain step, the sensor is located δz_s under the

environment, then the time it moves after impact (till next step) is $\delta t = \frac{\delta z_s}{\dot{z}_s}$. Using the velocity after impact ($\dot{\mathbf{z}}_0 = -e\dot{\mathbf{z}}_s$), part of the kinetic energy (after collision) is transferred to potential energy stored in the springs of the system. Using the conservation of energy and movement time, the velocity at the end of the integration step is calculated and the new location of the arm (above the environment) is found. The new position and velocity is then fed to the control low and to the integration routine. Other parameters that are being checked and updated are the maximum and minimum distances between the sensor and the arm that indicate true spring limitations. Figures 8.4 to 8.8 are selected outputs of the simulation, discussed in the next section.

8.3 CONCLUSIONS

A) While moving the arm in search pattern, for practical reasons, the velocity should be the maximum that still can locate desired locations. Figures 8.4 and 8.5 shows the same arm configuration with other plan velocities. For the surface shown, the higher speed (5 [mm/sec], causes the loss of much information.

B) Using a high stiffness configuration causes jittery movements of the arm and unstable contact with the environment (Fig. 8.5 and 8.7). The contact forces in the stiff arm are relatively high (Figures 8.6 - 8.8). This increases the friction forces in the $X-Y$ plan and sometimes can cause distractive contact with the environment. Complicated software has to be used in order to relate forces in the Z direction and the plane forces and to filter the friction forces (interference with the direction of edge information).

C) Another important advantage that is added by the compliance of the sensor, is its fast reactions to the opposite direction of the preload (faster then the reaction of the force control). When searching for a hole, the location of the hole is found by force information due to contact with the edges of the hole. For movements in the $X-Y$ plan in velocity of

2 [mm/sec], in extreme cases, when the tolerances between the peg and the hole are less then 0.056 [mm], there is good possibility that the force controller will not react at all. For other cases, with wider tolerances, the reaction (inserting the peg) can be too small for "good" contact that will alter and stop the search program due to force information. So, although compliance device reduces the bandwidth of the system, it is useful for most force controlled, assembly tasks.

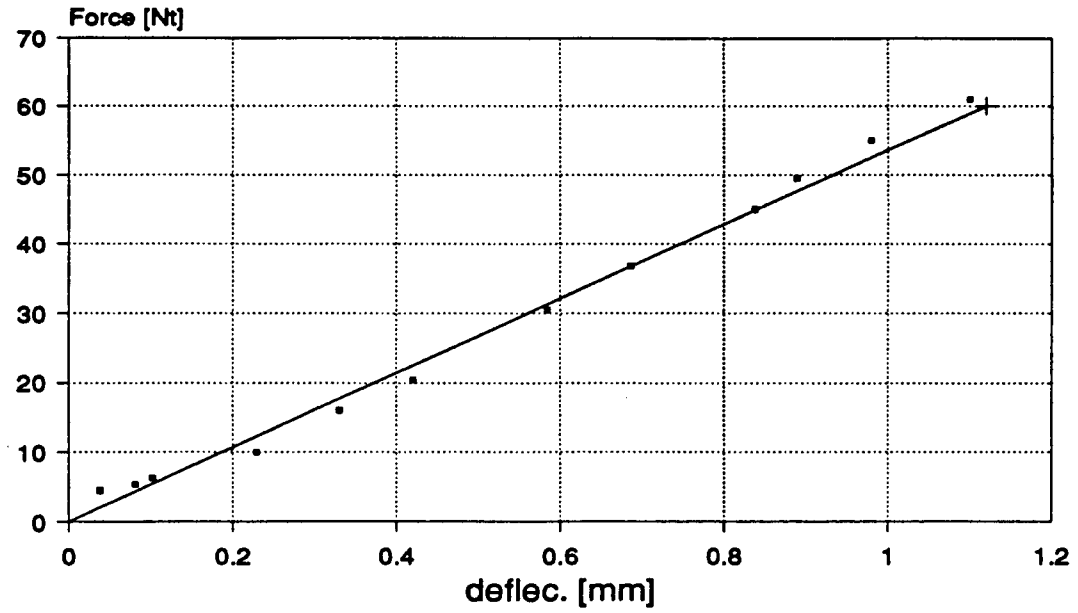

Figure 8.1: PUMA 560 measured arm stiffness.

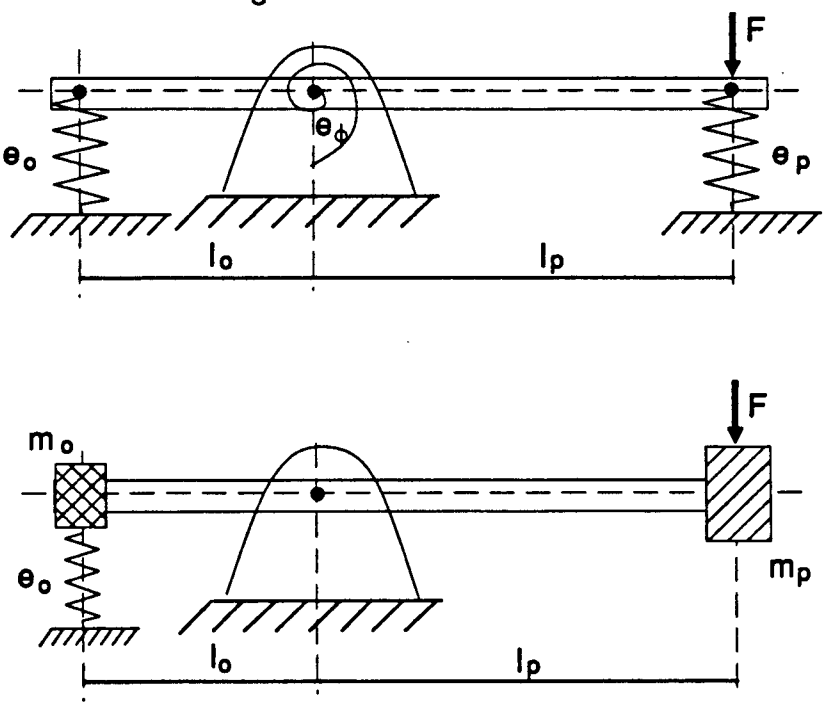

Figure 8.2: Reduction of translational compliance and mass to different location.

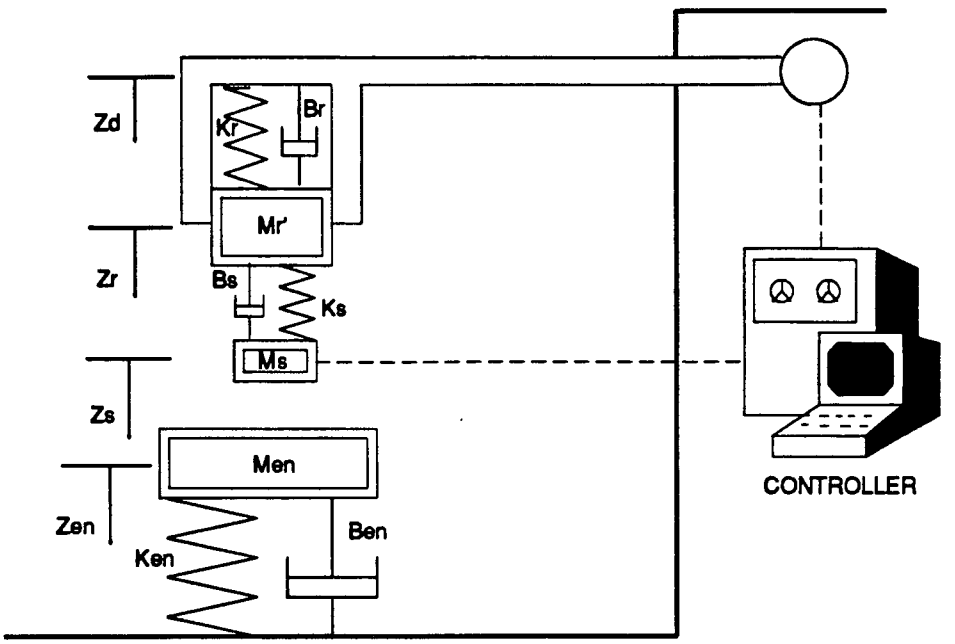

Figure 8.3: Linear model of the robot arm.

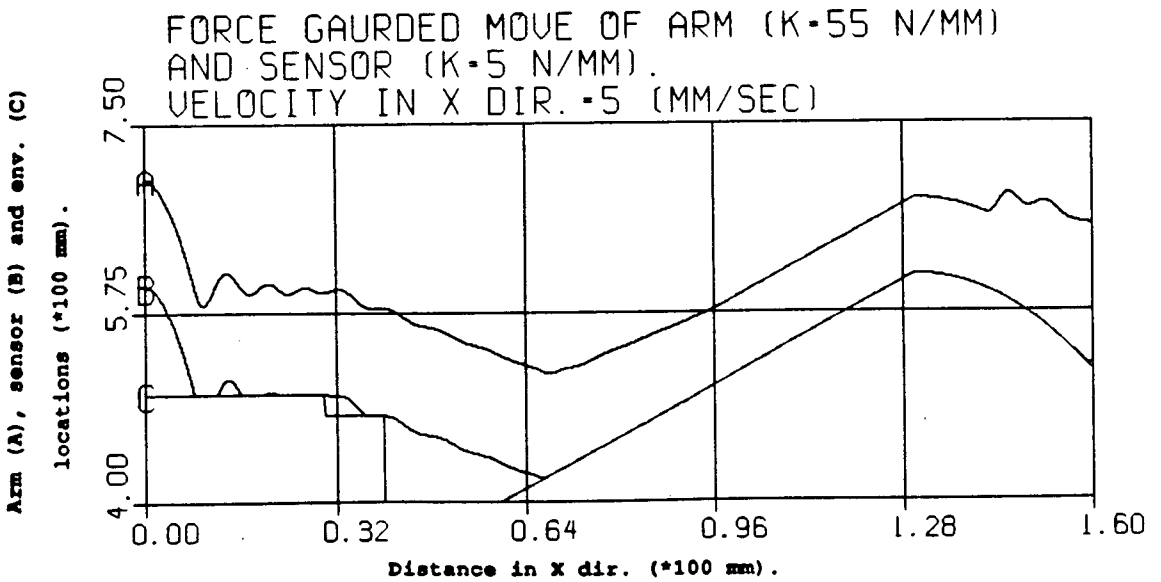

Figure 8.4: low stiffness sensor spring, high velocity.

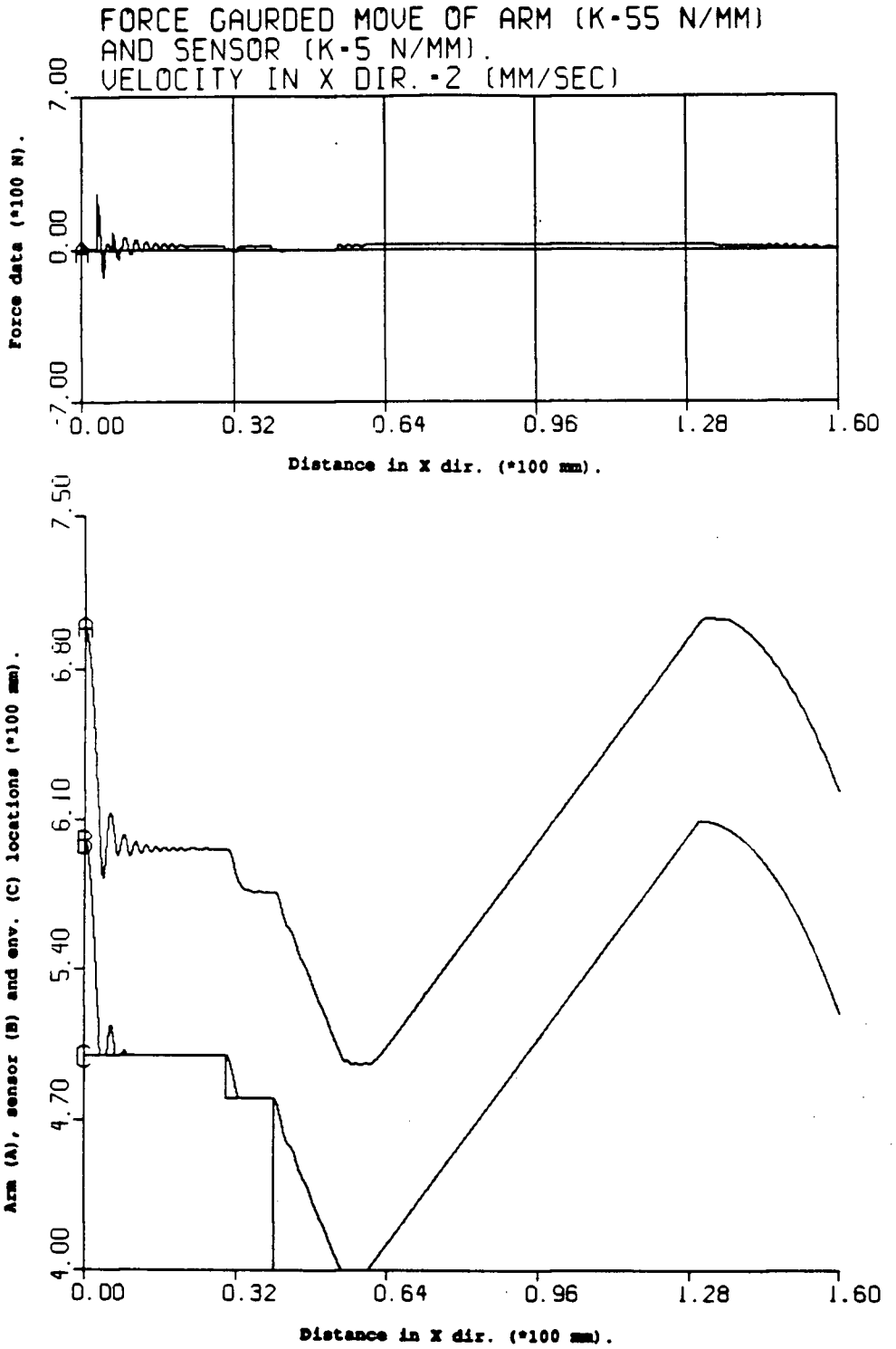

Figure 8.5: Low stiffness sensor spring, low velocity.

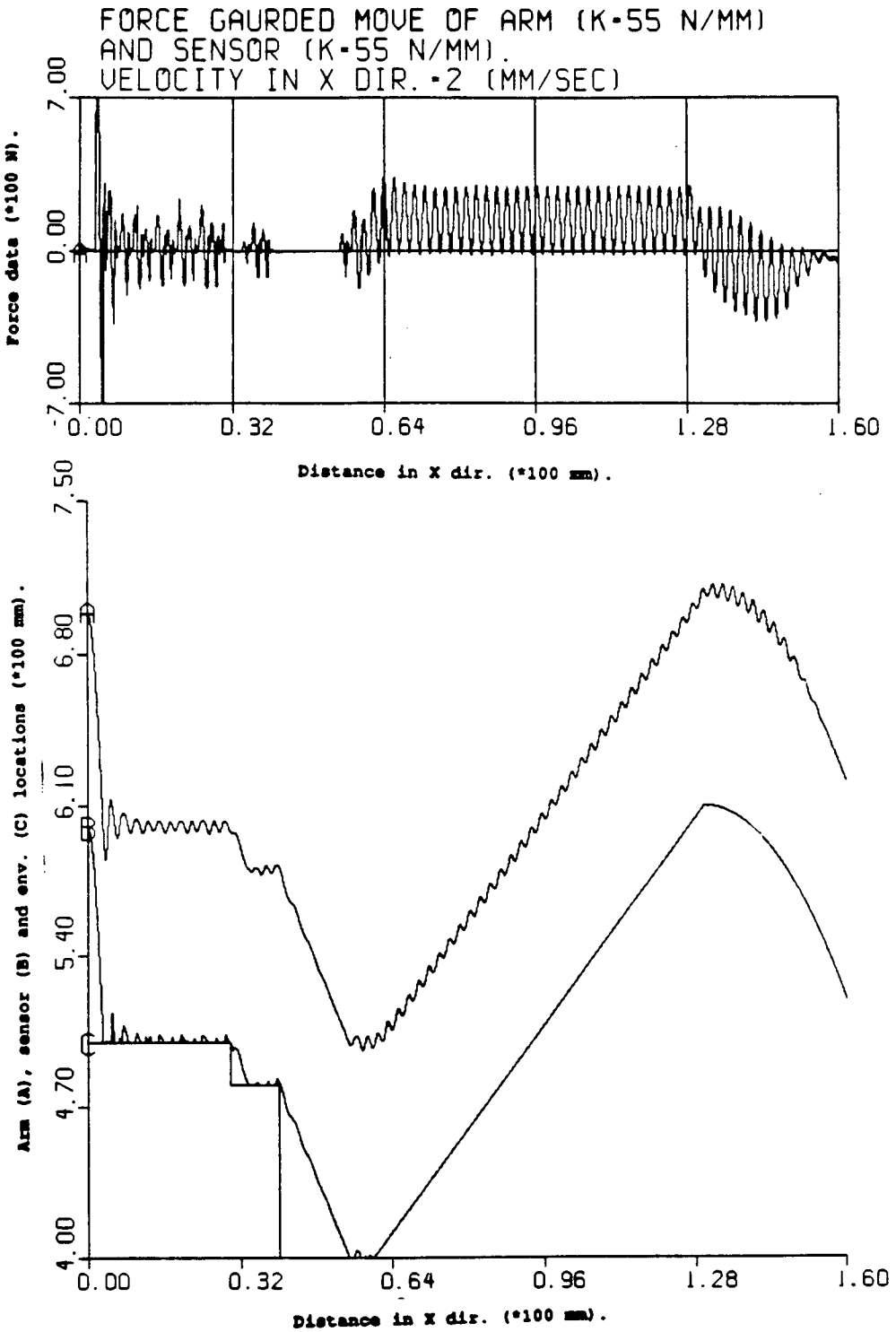

Figure 8.6: High stiffness sensor spring, low velocity.

Bibliography

[1] Asada H., Slotine J.J.E. 1986 by John Wiley & Sons, Inc.
Robot Analysis and Control.

[2] Albus J.S., Lumia R. and McCain H. *Hierarchical Control of Intelligent Machines Applied to Space Station Telerobots.* JPL Publication 87-13, Vol. II, July, 1987.

[3] Bihn D.G., Hsia T.C., July 1987. *A Universal Six-Joint Robot Controller.* JPL Publication 87-13, Vol. II.

[4] Brady M., Hollerbach J.M., Johnson T.L., Lozano-Perez T., Mason M.T. by The MIT Press (Series in Artificial Inteligence).
Robot Motion: Planning and Control. 1982, pp 567-585.

[5] Cherchas D.B., Boucher D.C. *Close Loop End Piece Control of Servo Controlled Manipulator.*

[6] Coiffet Phillip, *Interaction With the environment.* Robot Technology, Volume 2. 1983, pp 229-235.

[7] Cutkosly M.R., 1985 by Kluwer Academic Publishers.
Robotic Grasping and Fine Manipulation.

[8] deSilva C.W. *Control Sensors and Actuators.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[9] Drake S., Watson P., Simunovic S., 1977. *High Speed Robot Assembly of Precision*

Parts Using Compliance Instead of Sensory Feedback. Proc. 7th Int. Symp. on Industrial Robots.

[10] De Fazio T.L., Seltzer D.S., Whiteny D.E. *The IRCC Instrumented Remote Centre Compliance.* The Charles Stark Draper Lab. Inc., USA. Robot Sensor Vol. II. Tactile & Non-Vision. Edited by: Alan Pugh. IFS (Publications) Ltd. U.K., Springer Verlag, Berlin Heidelberg NewYork London Paris Tokyo 1986, pp 33-44.

[11] Eppinger D. Steven, Seering P. Warren, 1986 IEEE. *On Dynamic Models of Robot Force Control.* Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

[12] Feldmann K. and Classe D., *Sensor Aided Robot Programming.* Universitat Eriangen Nurnberg, West Gremany. Robot Vision and Sensory Control. October 1985.

[13] Fu K.S., Gonzalez R.C., Lee C.S.G. *Robotics Control, Sensing, Vision and Intelligence.* McGraw-Hill, Inc. 1987.

[14] Goto T., Inoyama T., Takeyasu K. *Pricise Insert Operation by Tectile Controlled Robot.* Proc. 2nd Confe. on Industrial Robot Technology. IFS (Publication) Ltd, Bedford, U.K., March 1974.

[15] Hunt K.H., *Kinnematic Geometry of Mechanisms.* Oxford University Press. 1978.

[16] Inoue H., *Force Feedback in Precise Assembly Tasks.* Memo No. 308, MIT Artificial Intelligence Lab., Cambridge, MA, USA.

[17] Katsuhiko O, *Discrete-Time Control Systems.* Prentice-Hall, INC., Englewood Cliffs, New Jersey 1987.

[18] Khatib O., Burdick J., *Motion and Force Control of Robot Manipulators.* Proceeding of the IEEE International conference on Robotics and Automation. April 1986.

[19] Lee T.T., and Shieh T.R., *Analysis and Compliance Control of a Multiple Insertion Assembly.* National Chiao Tung University, Taiwan. Robot Vision and Sensory Controls. October 1985.

[20] Lozano-Perez T., Mason M. T., Talor R. H., spring 1984. *Automatic Synthesis of Fine-Motion Strategies for Robots.* The International Journal of Robotics Reseearch. Vol. 3. No. 1.

[21] Maples J.A., Becker J.J., *Experiments in Force Control of Robotic Manipulators.* IEEE International conference on Robotics and Automation, Vol. 2, 1986.

[22] Martensson N., Johanson C., *Subassembly Of a Gearshaft By Industrial Robot.* Proc. 10th International Symp. on Industrial Robots. March 1980, Milan, Italy.

[23] Mason M. T., *Compliance and Force Control for Computer Controlled Manipulators.* IEEE transactions on System, Man and Cybernetics SMC-11, 6. June 1981.

[24] Mason M.T., Salisbury J.K., *Robot Hand and the Mechanics of Manipulatin.* The MIT Press series in artificial intelligence. 1985.

[25] Nevis J.L., Whiteny D.E.,*The Force Vector Assembly Concept.* Proc. 1st CISM Symp. Vol. II. Udine, Italy. 1975.

[26] O'Hara D.D., *Multiprocessor Robot Control For Assembly: A Demonstration With Tactile Feedback For Prismatic Shape Block Insertion.* Proc. Int. Conf. Robot and Sensory Controls, February 1988.

[27] Ohwovoriole M.S., Hill J.W., Roth B., *On the Theory of Single and Multiple Insertions in Industrial assemblies.* Proc. 10th International Symp. on Industrial Robots. March 1980, Milan, Italy.

[28] Paul R.P., *Problems and Research Issues Associated With the Hhbrid Control of Force and Displacement.* JPL Publication, Vol. II, July 1987.

[29] Paul R.P., Shimano B., *Compliance and Control.* Proceedings of the 1976 Joint Automatic Control Conference.

[30] Raibert M.H., Craig J.J., *Hybrid Position/Force Control of Manipulators.* Trans. ASME Journal of Dynamics, Systems, Measurements, and Control, Vol. 102, June 1981.

[31] Rivin. E.I. *Mechanical Design of Robots.* McGraw-Hill Inc., 1988.

[32] Roberts R.K., Paul R.P., Hillberry B.M., *The Effect of Wrist Force Sensor Stiffness on the Control of Robot Manipulators.* Proceeding of the IEEE International Conference on Robotics and Automation. April 1985.

[33] Simunovic S., 1975. *Force Information in Assembly Proccesses.* Proc. 5th Int. Symp. on Industrial Robots.

[34] Salisbury J.K., *Active Stiffness Control of a Manipulator in Cartesian Coordinates.* Proc. 19th IEEE Conf. on Deci. and Contr., 1980, pp. 95-100.

[35] Saridis G.N. and Valavanis K.P., *Software and Hardware for Intelligent Robots.* JPL Publication 87-13, Vol. II, July, 1987.

[36] Shigley J.E., *Mechanical Engineering Design.* McGraw-Hill Inc., 1977.

[37] Stepien T.M. et al. *Control of Tool/Workpiece Contact Forces With Application to Robotic Deburing.* Proc. IEEE Conf. on Robotic and Automation. St. Louis Mo., 1985.

[38] Unimation Incorporated , A Westinghouse Company.
Unimate Puma Robot Manual (398H). August 1984.
500 Series Equipment Manual for VAL II and VAL PLUS Operating Systems (398U1). March 1985.
Programing Manual User's Guide to VAL II (398T1). August 1984.

[39] vanBrussel H., Simons J.,*Automatic Assemblly By Active Force Feedback Accommodation.* Robot Sensor Vol. II. Tactile & Non-Vision. Edited by: Alan Rugh. IFS (Publications) Ltd. U.K., 1986.

[40] vanBrussel H., Belien H. and Thielemans H., *Force Sensing for Advanced Robot Control.* Katholieke Universiteit Leuven, Belgium. Robot Vision and Sensory Control. October 1985.

[41] Vukobratovic M., *Introduction to Robotics.* Springer Verlag Berlin Heidelberg NewYork London Paris Tokyo 1989.

[42] Vukobratovic M., Potkonjak V., *Applied Dynamics and CAD of Manipulation Robots.* Scientific Fundamentals of Robotics 6. Springer-Verlag Berlin Heidelberg NewYork London Paris Tokyo 1989.

[43] Waston P.C., *A Multidimensional System Analysis of the Assembly Process as Performed by a Manipulator.* 1st North Amer. Robot Conf. Chicago, 1976.

[44] West H., Asada H., *A Method for the Design of Hybrid Position/Force Controller foe Manipulators Constraind by Contact With the Environment.* Proc. 1985 IEEE

Conf. Robotics and Automation 251-259.

[45] Whitney E.D., June 1977. *Force Feedback Control of Manipulator Fine Motions.* Jurnal of Dynamic Systems, Mesurement, and Control.

[46] Whiteny E.D., March 1982. *Quasi-Static Assembly of Compliantly Supported Rigid Parts.* Trans. ASME Journal of Dynamics, Systems, Measurement, and Control. March 1982.

[47] Whiteny E.D., 1987. *Historical perspective and State of the Art in Robot Force Control.* The International Journal of Robotics Research. Vol. 6. No. 1.

[48] Zhang H., Paul R.P., *Hybrid Control of Robot Manipulators.* 1985 IEEE.

Appendix A

MAIN TASK and SUBTASKS:

A.1 main program PUMP:

```
 1 ; main task PUMP (May. 24 /Kotzev)
 2 ;
 3         ATTACH
 4         CALL gear1.ini1
 5         CALL pick
 6         CALL gear1.ini2
 7         CALL insert
 8         CALL gear2.ini1
 9         CALL pick
10         CALL gear2.ini2
11         CALL insert
12         CALL top.ini1
13         CALL pick
14         CALL top.ini2
15         CALL insert
16         CALL screw1.ini1
17         CALL pick
18         CALL screw1.ini2
19         CALL place
20         CALL repeat.bolt
21         CALL tool.ini1
22         CALL pick
23         CALL tool.ini2
24         CALL screw
25         CALL repeat.screw
26         CALL tool.ini3
27         CALL place
28         CALL pcstop
```

A.2 subtask INSERT:

```
 1 ; subtask INSERT (May 25 / Kotzev)
 2 ;
 3         TYPE "********  INSERT started "
```

```
 4          CALL move
 5          CALL approach
 6          IF search.pc == 1 GOTO 10
 7          CALL search
 8      10  CALL comply
 9          CALL clear
10          TYPE "********  INSERT ended "
11          RETURN
```

A.3 subtask PICK:

```
 1 ; subtask PICK (Jan.12 /Kotzev)
 2 ;
 3          TYPE "******** PICK started "
 4          CALL move
 5          CALL approach
 6          CALL grasp
 7          CALL clear
 8          TYPE "******** PICK ended "
 9          RETURN
```

A.4 subtask PLACE:

```
 1 ; subtask PLACE (June 6 - Kotzev)
 2 ;
 3          TYPE "********  PLACE started "
 4          CALL move
 5          CALL approach
 6          OPEN
 7          CALL clear
 8          TYPE "********  PLACE ended "
 9          RETURN
```

A.5 subtask SCREW:

```
 1 ; subtask SCREW (June 18 - Kotzev)
 2 ;
 3          TYPE "********  SCREW started "
 4          CALL move
 5          CALL approach
 6          CALL rotate
 7          CALL clear
 8          TYPE "********  SCREW ended "
 9          RETURN
```

Appendix B

SUBROUTINES and SERVICE PROGRAMS:

B.1 program APPROACH:

```
 1 ; subroutine APPROACH (June 21 - Kotzev)
 2         CALL pcstop
 3         TYPE "  ++++++++  APPROACH started"
 4 ; Weight the end effector equipment:
 5         IF weight.ex == 1 THEN
 6             CLOSEI
 7             TIMER (1) = 0
 8             WAIT TIMER(1) > 1
 9             IF null.ex == 1 THEN
10                 CALL null.sensor
11                 IF null.fail == 1 GOTO 5
12                 GOTO 25
13             END
14             mass.before = weight
15    25       TYPE "-------- f.m[3] = ", /F6.2, mass.before, "[Kg]"
16         END
17         IF open.grip == 1 THEN
18             OPENI
19         END
20 ; Move in X Y plane and on Z axis to temp.pnt (shifting point):
21         SPEED speed.high  ALWAYS
22         SET temp.pnt = SHIFT(HERE BY shift.x, shift.y, 0)
23         MOVES temp.pnt
24         DECOMPOSE a[] = tool.pnt
25         sx = a[0]+shift.x
26         sy = a[1]+shift.y
27         sz = a[2]+shift.z
28         SET temp.pnt = TRANS(sx, sy, sz, a[3], a[4], a[5])
29         MOVES temp.pnt
30         BREAK
31         IF open.grip == 1 THEN
32             SPEED speed.appro  ALWAYS
33             MOVES tool.pnt
34             GOTO 100
35         END
```

```
36 ; Move from the shift point to contact:
37         TYPE "    --------  APPROACH.PC started "
38         PCEXECUTE approach.pc, -1, 0
39         ALTER (-1, 18, , 1)
40         SPEED speed.appro  ALWAYS
41         IF shift.x > 0 THEN
42             delta.x = delta.shift
43         END
44         IF shift.y > 0 THEN
45             delta.y = delta.shift
46         END
47         IF shift.z > 0 THEN
48             delta.z = delta.shift
49         END
50     10  SET point.appro = SHIFT(HERE BY -delta.x, -delta.y, -delta.z)
51         count = count+1
52         shift.max = shift.x+shift.y+shift.z+shift.over
53         IF (count*delta.shift) > (shift.max) THEN
54             IF position.appro == 1 THEN
55                 search.pc = 1
56                 GOTO 15
57             END
58         END
59         MOVES point.appro
60         BREAK
61         IF SIG(2025) THEN
62             IF force.appro == 1 THEN
63                 IF rot.appro == 1 THEN
64                     count.rot = count.rot+1
65                     TYPE count.rot
66                     IF count.rot > 5 GOTO 15
67                     SIGNAL -2025
68                     HERE temp.pnt
69                     DECOMPOSE a[] = temp.pnt
70                     r[5] = 5*count.rot
71                     SET temp.pnt = TRANS(a[0], a[1], a[2], a[3], a[4], 5)
72                     MOVES temp.pnt
73                     GOTO 500
74                 END
75                 GOTO 15
76             END
77         END
78    500  GOTO 10
79 ;
80     15  CALL pcstop
```

```
81          TYPE "    ========  f[3] = ", /F6.2, f[3], "[Kg]"
82     100  TYPE "  ++++++++  APPROACH finished "
83          RETURN
```

```
 1 ; APPROACH.PAR (June 5 - Kotzev)
 2 ;
 3          second = 0
 4          count = 0
 5          count.rot = 0
 6          rot.appro = 0
 7          search.pc = 0
 8          weight.ex = 1
 9          null.ex = 1
10          search.pc = 0
11          open.grip = 1
12          force.appro = 1
13          position.appro = 1
14          shift.x = 0
15          shift.y = 0
16          shift.z = 5
17          delta.shift = 0.3
18          shift.over = 5
19          delta.x = 0
20          delta.y = 0
21          delta.z = 0
22          speed.high = 50
23          speed.appro = 10
24          appro.max[1] = 2;      Forces during approach.
25          appro.max[2] = 2
26          appro.max[3] = 2.75
```

```
 1 ; program APPROACH.PC (June 5 - Kotzev)
 2 ;
 3          IF SIG(2020) == 0 THEN
 4              TYPE "    --------  APPROACH.PC ended "
 5              HALT
 6          END
 7 ;
 8 ; Check forces in X, Y and Z directions:
 9          CALL jr3.dat
10          FOR pc = 1 TO 3
11              IF ABS(f.m[pc]) > appro.max[pc] THEN
12                  SIGNAL 2025
```

```
13              END
14          END
15          ALTOUT 0, 0, 0, 0, 0, 0, 0
```

B.2 program CLEAR:

```
 1 ; subroutine CLEAR (June 21 - Kotzev)
 2 ;
 3          CALL pcstop
 4          TYPE "  ++++++++  CLEAR started "
 5          SPEED speed.clear  ALWAYS
 6          IF clear.grip == 1 THEN
 7              OPEN
 8              TIMER (1) = 0
 9              WAIT TIMER(1) > 0
10          END
11          IF second == 1 THEN
12              SET temp.pnt = SHIFT(HERE BY clear.x, clear.y, clear.z)
13              MOVES temp.pnt
14              BREAK
15          END
16          DECOMPOSE b[] = tool.pnt
17          HERE temp.pnt
18          DECOMPOSE a[] = temp.pnt
19          MOVES TRANS(a[0], a[1], b[2]+safe.z, a[3], a[4], a[5])
20          BREAK
21          IF clear.rotate == 1 THEN
22              SPEED speed.rotate  ALWAYS
23              HERE #temp.pnt
24              DECOMPOSE a[] = #temp.pnt
25              MOVE #PPOINT(a[0], a[1], a[2], a[3], a[4], clear[5])
26              BREAK
27          END
28          TYPE "  ++++++++  CLEAR ended "
29          RETURN
```

```
 1 ; CLEAR.PAR (July 21 - Kotzev)
 2 ;
 3          speed.clear = 100
 4          clear.rotate = 0
 5          clear[5] = -260
 6          clear.grip = 0
 7          clear.x = 0
 8          clear.y = 0
```

```
 9          clear.z = 0
10          RETURN
```

B.3 program COMPLY:

```
 1 ; subroutine COMPLY (August 29 - Kotzev)
 2 ;
 3          CALL pcstop
 4          TYPE "  ++++++++  COMPLY started "
 5 ; t[] was decomposed to be the target point (done by the
 6 ; initialization program).
 7          TYPE "    --------  COMPLY.PC started "
 8          PCEXECUTE comply.pc, -1, 1
 9          ALTER (-1, 17, , 1)
10 ; Rotate if rotate==1:
11     10   DELAY delaytime
12          IF SIG(2025) THEN
13              IF comply.force == 1 THEN
14                  HERE temp.pnt
15                  DECOMPOSE aa[] = temp.pnt
16                  IF ABS(aa[2]-t[2]) > delcomply.z GOTO 20
17              END
18          END
19          HERE temp.pnt
20          DECOMPOSE aa[] = temp.pnt
21          IF ABS(aa[2]-t[2]) > (lower.pnt-3) GOTO 20
22          GOTO 10
23 ;
24     20   CALL pcstop
25          TYPE "  ++++++++  COMPLY ended "
26          RETURN
```

```
 1 ;COMPLY.PAR (August 28 - Kotzev)
 2 ;
 3          delaytime = 1
 4          rotate = 0
 5          comply.force = 0
 6          mid.pnt = 50
 7          delcomply.z = 50
 8          lower.pnt = 100
 9          f.max[1] = 10; information to stop comply.
10          f.max[2] = 10
11          f.max[3] = 3
12          f.max[4] = 1000
```

```
13          f.max[5] = 1000
14          f.max[6] = 250
15          comax[1] = 0.5
16          comin[1] = -0.5
17          comax[2] = 0.5
18          comin[2] = -0.5
19          comax[3] = 0.5
20          comin[3] = -0.5
21          comax[4] = 30
22          comin[4] = -30
23          comax[5] = 30
24          comin[5] = -30
25          comax[6] = 30
26          comin[6] = -30
27          gain[1] = 10
28          gain[2] = 10
29          gain[3] = 10
30          gain[4] = 0.5
31          gain[5] = 0.5
32          gain[6] = 1
33          del.max[1] = 30
34          del.max[2] = 30
35          del.max[3] = 30
36          del.max[4] = 20
37          del.max[5] = 20
38          del.max[6] = 50
```

```
 1 ; COMPLY.PC (Aug. 3 - Kotzev)
 2 ;
 3          IF SIG(2020) == 0 THEN
 4              TYPE "    --------  COMPLY.PC ended "
 5              HALT
 6          END
 7          HERE temp.pnt
 8          DECOMPOSE a[] = temp.pnt
 9          IF ABS(b[2]-a[2]) > lower.pnt THEN
10              SIGNAL 2026
11          END
12          CALL jr3.dat
13 ; Monitor forces to stop the program:
14          IF ABS(f.m[3]) > ABS(f.max[3]) THEN
15              SIGNAL 2025
16          END
17 ; Initiate fsign[]:
```

```
18          FOR pc = 1 TO 6
19              fsign[pc] = 0
20          END
21  ; Monitor forces in X dir.:
22          IF f.m[1] > comax[1] THEN
23              fsign[1] = -ABS(f.m[1]-comax[1])
24          END
25          IF f.m[1] < comin[1] THEN
26              fsign[1] = ABS(f.m[1]-comin[1])
27          END
28  ; Monitor forces in Y dir.:
29          IF f.m[2] > comax[2] THEN
30              fsign[2] = -ABS(f.m[2]-comax[2])
31          END
32          IF f.m[2] < comin[2] THEN
33              fsign[2] = ABS(f.m[2]-comin[2])
34          END
35  ; Monitor forces in Z dir.:
36          IF f.m[3] > comax[3] THEN
37              fsign[3] = ABS(f.m[3]-comax[3])
38          END
39          IF f.m[3] < comin[3] THEN
40              fsign[3] = -ABS(f.m[3]-comin[3])
41          END
42  ;Monitor torques around X:
43          IF f.m[4] > comax[4] THEN
44              fsign[4] = -ABS(f.m[4]-comax[4])
45          END
46          IF f.m[4] < comin[4] THEN
47              fsign[4] = ABS(f.m[4]-comin[4])
48          END
49  ; Monitor torques around Y:
50          IF f.m[5] > comax[5] THEN
51              fsign[5] = -ABS(f.m[5]-comax[5])
52          END
53          IF f.m[5] < comin[5] THEN
54              fsign[5] = ABS(f.m[5]-comin[5])
55          END
56  ; Monitor torque around Z:
57          IF f.m[6] > comax[6] THEN
58              fsign[6] = ABS(f.m[6]-comax[6])
59          END
60          IF f.m[6] < comin[6] THEN
61              fsign[6] = -ABS(f.m[6]-comin[6])
62          END
```

```
63 ;
64          FOR pc = 1 TO 6
65              del[pc] = fsign[pc]*gain[pc]
66              IF ABS(del[pc]) > del.max[pc] THEN
67                  del[pc] = del.max[pc]*fsign[pc]/ABS(fsign[pc])
68              END
69          END
70          ALTOUT 0, del[1], del[2], del[3], del[4], del[5], del[6]
```

B.4 program COMSCREW:

```
 1 ; subroutine COMPLY.SCREW (June 21 - Kotzev)
 2 ;
 3          CALL pcstop
 4          TYPE "  ++++++++  COMSCREW started "
 5          TYPE "    --------  COMPLY.PC started "
 6      50  PCEXECUTE comply.pc, -1, 1
 7          ALTER (-1, 17, , 1)
 8          HERE #temp.pnt
 9          DECOMPOSE b[] = #temp.pnt
10      10  HERE #temp.pnt
11          DECOMPOSE a[] = #temp.pnt
12          IF a[5] > 0 GOTO 20
13          IF ABS(b[5])-ABS(a[5]) > 60 THEN
14              CALL pcstop
15              GOTO 50
16          END
17          DELAY delaytime
18          IF SIG(2025) GOTO 20
19          TYPE del[1], del[2], del[3], del[4], del[5], del[6]
20          GOTO 10
21 ;
22      20  CALL pcstop
23          TYPE "  ++++++++  COMPLY ended "
24          RETURN
```

```
1 ; COMSCREW.PAR (July 21 - Kotzev)
2 ;
3          comply.force = 1
4          comax[3] = -0.6
5          comin[3] = -0.5
6          comax[6] = -250
7          comin[6] = -240
8          del.max[1] = 0
```

```
 9         del.max[2] = 0
10         del.max[3] = 3
11         del.max[4] = 0
12         del.max[5] = 0
13         del.max[6] = 250
```

B.5 GEAR1, GEAR2 parameters:

```
 1 ; program GEAR1.INI1 (May. 24 /Kotzev)
 2 ;
 3         CALL pcstop
 4         TYPE "  "
 5         TYPE " GEAR1 ASSEMBLY "
 6         TYPE "  "
 7 ; CONFIGURATION:
 8         z.sensor = 31
 9         z.rcc = 43
10         z.gripper = 112
11         TOOL TRANS(0, 0, z.sensor+z.rcc+z.gripper, 90, -90, 45)
12         mass = 0.17
13         ds[1] = 8
14         ds[2] = 8
15         z.tool = 57
16         ds[3] = z.rcc+z.gripper+z.tool
17 ; MOVE parameters:
18         SET tool.pnt = gear1.pnt1
19         safe.z = 300
20         speed.move = 100
21         zero.position = 0
22 ; APPROACH parameters:
23         CALL approach.par
24 ; GRASP parameters:
25         temp.force = 10
26         speed.grasp = 10
27         weight.z = 30
28 ; CLEAR parameters:
29         CALL clear.par
```

```
 1 ; program GEAR1.INI2 (Aug. 3 - Kotzev)
 2 ;
 3         CALL pcstop
 4 ; CONFIGURATION:
 5         z.tool = 57
 6         TOOL TRANS(0, 0, z.sensor+z.rcc+z.gripper+z.tool, 90, -90, 45)
```

```
 7         mass = 0.17
 8         ds[1] = 8
 9         ds[2] = 8
10         ds[3] = z.rcc+z.gripper+z.tool
11 ; MOVE parameters:
12         SET tool.pnt = gear1.pnt2
13         safe.z = 300
14         speed.move = 100
15 ; APPROACH parameters:
16         CALL approach.par
17         weight.ex = 0
18         open.grip = 0
19         shift.over = 10
20 ; SEARCH parameters:
21         CALL search.par
22         speed.xy = 2
23         delx = 0.3
24         dely = 0.3
25         delsearch.z = 3
26 ; COMPLY parameters:
27         CALL comply.par
28         comply.force = 1
29         f.max[2] = 2.5
30         delcomply.z = 10
31         lower.pnt = 31
32         speed.comply = 5
33         comax[3] = -0.6
34         comin[3] = -0.5
35 ; CLEAR parameters:
36         CALL clear.par
37         clear.grip = 1
```

```
 1 ; GEAR2.INI1 (August 28 - Kotzev)
 2 ;
 3         CALL pcstop
 4         TYPE "  "
 5         TYPE " GEAR2 ASSEMBLY "
 6         TYPE "  "
 7 ; CONFIGURATION:
 8         DECOMPOSE t[] = gear1.pnt1
 9         SET gear2.pnt1 = TRANS(t[0], t[1]+43, t[2], t[3], t[4], t[5])
10         z.sensor = 31
11         z.rcc = 43
12         z.gripper = 112
```

```
13          TOOL TRANS(0, 0, z.sensor+z.rcc+z.gripper, 90, -90, 45)
14          mass = 0.17
15          ds[1] = 8
16          ds[2] = 8
17          z.tool = 97
18          ds[3] = z.rcc+z.gripper+z.tool
19  ; MOVE parameters:
20          SET tool.pnt = gear2.pnt1
21          safe.z = 300
22          speed.move = 100
23          zero.position = 0
24  ; APPROACH parameters:
25          CALL approach.par
26  ; GRASP parameters:
27          temp.force = 10
28          speed.grasp = 10
29          weight.z = 70
30  ; CLEAR parameters:
31          CALL clear.par
```

```
 1  ; GEAR2.INI2 (August 28 - Kotzev)
 2  ;
 3          CALL pcstop
 4  ; CONFIGURATION:
 5          DECOMPOSE t[] = gear1.pnt2
 6          SET gear2.pnt2 = TRANS(t[0], t[1]+30, t[2], t[3], t[4], t[5])
 7          z.tool = 97
 8          TOOL TRANS(0, 0, z.sensor+z.rcc+z.gripper+z.tool, 90, -90, 45)
 9          mass = 0.17
10          ds[1] = 8
11          ds[2] = 8
12          ds[3] = z.rcc+z.gripper+z.tool
13  ; MOVE parameters:
14          SET tool.pnt = gear2.pnt2
15          safe.z = 300
16          speed.move = 100
17  ; APPROACH parameters:
18          CALL approach.par
19          weight.ex = 0
20          open.grip = 0
21          shift.over = 10
22          position.appro = 1
23  ; SEARCH parameters:
24          CALL search.par
```

```
25 ; COMPLY parameters:
26         CALL comply.par
27         comply.force = 1
28         lower.pnt = 63
29         delcomply.z = 63
30         comax[3] = -1.2
31         comin[3] = -1.1
32         comax[6] = 25
33         comin[6] = 20
34         gain[3] = 10
35         del.max[3] = 62
36 ; CLEAR parameters:
37         CALL clear.par
38         clear.grip = 1
```

B.6 program GRASP:

```
 1 ; program GRASP (Feb.5 /Kotzev)
 2         CALL pcstop
 3         TYPE "  ++++++++  GRASP started "
 4         IF open.grip == 1 THEN
 5             CLOSE
 6         END
 7         IF temp.force > 0 THEN
 8             DO
 9                 CALL jr3.dat
10                 APPRO HERE, -0.5
11             UNTIL ABS(f.m[3]) > temp.force
12         END
13         IF weight.ex == 1 THEN
14             SPEED speed.grasp  ALWAYS
15             DEPARTS weight.z
16             BREAK
17             TIMER (1) = 0
18             WAIT TIMER(1) > 2
19             CALL weight
20             mass.after = weight
21             TYPE "--------  f.m[3] = ", /F6.2, mass.after, "[Kg]"
22             temp.mass = ABS(mass.after-mass.before)
23             IF ABS(temp.mass) < 0.7*mass THEN
24                 TYPE " ERROR - failure to grasp the tool "
25             ELSE
26                 TYPE "-------- grasped the tool "
27             END
28         END
```

```
29          TYPE "  ++++++++  GRASP ended "
30          RETURN


 1 ; program GRASP.SUB
 2 ;~~~~~~~~~~~~~~~~~~~~Last modification 28 - 9 -   5
 3 ; PARameters: CLOSE.GAP = to face the peg's sholders.
 4 ;
 5          PCEND
 6          close.gap = -2
 7          TYPE "    --------  GRASP.SUB started "
 8          SPEED speed.grasp.sub  ALWAYS
 9          DECOMPOSE t[] = work.point
10          SET temp = TRANS(t[0], t[1], t[2]+30, t[3], t[4], t[5])
11          MOVE temp
12          BREAK
13          SIGNAL 1
14          TIMER (1) = 0
15          WAIT TIMER(1) > 1
16          CALL weight
17          masstool1.0 = weight
18          TYPE "--------  f.m[3] = ", /F6.2, masstool1.0, "[Kg]"
19          OPEN
20          MOVE work.point
21          BREAK
22          CLOSE
23          TIMER (1) = 0
24          WAIT TIMER(1) > 1
25          APPRO work.point, close.gap
26          BREAK
27          TIMER (1) = 0
28          WAIT TIMER(1) > 1
29          MOVE temp
30          BREAK
31          CALL weight
32          masstool1.1 = weight
33          TYPE "--------  f.m[3] = ", /F6.2, masstool1.1, "[Kg]"
34          tool1.mass = masstool1.1-masstool1.0
35          IF ABS(tool1.mass) < 0.1 THEN
36              TYPE "  Error - failure to grasp the tool "
37              OPEN
38              MOVE SHIFT(HERE BY 0, 0, 0.1)
39              flag.tool1 = 0
40          ELSE
41              TYPE "--------  grasped the tool"
```

```
42              flag.tool1 = 1
43          END
44          TYPE "     --------  GRASP.SUB ended "
45          RETURN
```

B.7 program JR3.DAT:

```
 1 ; program JR3.DAT (Jan.10 /Kotzev)
 2 ;
 3 ; The program is based on GET.DATA to recieve force/moment
 4 ; information at the tool's tip in [Kg] and [Kg*mm] units.
 5 ; in order to receive the correct data, TOOL trans. must be
 6 ; included in the prog. i.e: TOOL TRANS(rx, ry, rz, 0, -90, 0)
 7 ; This includes the 90 degees shift in the JR3 force sensor.
 8 ;
 9          start.err = 0
10          mask.err = 0
11          sat.err = 0
12          check.err = 0
13          trailer.err = 0
14 ;
15          fmax[1] = 25
16          fmax[2] = 25
17          fmax[3] = 50
18          fmax[4] = 75
19          fmax[5] = 75
20          fmax[6] = 75
21 ;
22 ;CHECKING THE "START" WORD:
23      1   start = IOGET(-1028)
24          IF start <> 0 THEN
25              start.err = start.err+1
26              IF start.err > 10 THEN
27                  TYPE "  JR3 error - problems with the START word"
28                  GOTO 500
29              END
30              WAIT
31              GOTO 1
32          END
33 ;
34 ;CHECKING THE "MASK" WORD:
35      2   mask = IOGET(-1028)
36          IF mask BAND ^40377 <> ^356 THEN
37              IF mask BAND ^377 <> ^356 THEN
38                  mask.err = mask.err+1
```

```
39                  IF mask.err > 10 THEN
40                      TYPE "  JR3 error - problems with the MASK word"
41                      GOTO 500
42                  END
43                  GOTO 1
44              END
45              IF mask BAND ^40000 THEN
46                  sat.err = sat.err+1
47                  IF sat.err > 10 THEN
48                      TYPE "  JR3 error - SATURATION error"
49                      GOTO 500
50                  END
51                  GOTO 1
52              END
53          END
54 ;
55          z.d1 = IOGET(-1028)
56          z.d2 = IOGET(-1028)
57          z.d3 = IOGET(-1028)
58          z.d4 = IOGET(-1028)
59          z.d5 = IOGET(-1028)
60          z.d6 = IOGET(-1028)
61 ;
62 ; Cheking the CHECK word (check-sum):
63      3   check = IOGET(-1028)
64          IF (z.d1+z.d2+z.d3+z.d4+z.d5+z.d6+check) <> 0 THEN
65              check.err = check.err+1
66              IF check.err > 10 THEN
67                  TYPE " JR3 error - CHECK word incorrect"
68                  GOTO 500
69              END
70              GOTO 1
71          END
72 ;
73 ; Checking the TRAILER word:
74      4   trailer = IOGET(-1028)
75          IF trailer <> ^100000 THEN
76              trailer.err = trailer.err+1
77              IF trailer.err > 10 THEN
78                  TYPE " JR3 error - TRAILER word incorrect"
79                  GOTO 500
80              END
81              GOTO 1
82          END
83 ; Forces in [lb] and torques in [lb-in]:
```

```
 84          f[1] = z.d1*.122E-01
 85          f[2] = -z.d2*.122E-01
 86          f[3] = z.d3*.244E-01
 87          f[4] = z.d4*.336E-01
 88          f[5] = -z.d5*.336E-01
 89          f[6] = z.d6*.336E-01
 90 ;
 91          FOR jr = 1 TO 6
 92              IF ABS(f[jr]) > fmax[jr] THEN
 93                  TYPE "JR3 error - OVERLOAD"
 94              END
 95          END
 96 ; Forces in [Kg] and Torques in [Kg-mm]:
 97          f.m[1] = 0.445*f[1]
 98          f.m[2] = 0.445*f[2]
 99          f.m[3] = 0.445*f[3]
100          f.m[4] = 11.3*f[4]+ds[3]*f.m[2]-ds[2]*f.m[3]
101          f.m[5] = 11.3*f[5]-ds[3]*f.m[1]+ds[1]*f.m[3]
102          f.m[6] = 11.3*f[6]+ds[2]*f.m[1]-ds[1]*f.m[2]
103    500   RETURN
```

B.8 program MOVE:

```
  1 ; subroutine MOVE (June 22 - Kotzev)
  2 ;
  3          CALL pcstop
  4          TYPE "  ++++++++  MOVE started "
  5          SPEED speed.move  ALWAYS
  6          DECOMPOSE b[] = tool.pnt
  7          HERE temp.pnt
  8          DECOMPOSE a[] = temp.pnt
  9          IF a[2]-safe.z > b[2] THEN
 10              MOVES TRANS(b[0], b[1], a[2], b[3], b[4], b[5])
 11              BREAK
 12              MOVES TRANS(b[0], b[1], b[2]+safe.z, b[3], b[4], b[5])
 13              BREAK
 14          ELSE
 15              MOVES TRANS(a[0], a[1], b[2]+safe.z, a[3], a[4], a[5])
 16              BREAK
 17              MOVES TRANS(b[0], b[1], b[2]+safe.z, b[3], b[4], b[5])
 18              BREAK
 19          END
 20          IF zero.position == 1 THEN
 21              HERE #temp.pnt
 22              DECOMPOSE a[] = #temp.pnt
```

```
23              MOVE #PPOINT(a[0], a[1], a[2], a[3], a[4], start[5])
24              BREAK
25          END
26          TYPE "  ++++++++  MOVE ended "
27          RETURN
```

B.9 program NULL.SENSOR:

```
 1 ; Null.sensor (June 17 - Kotzev)
 2 ;
 3          count = 0
 4          null.fail = 0
 5      10  SIGNAL 1
 6          count = count+1
 7          IF count > 10 THEN
 8              null.fail = 1
 9              TYPE "-----  can't reset force data! "
10              GOTO 20
11          END
12          CALL jr3.dat
13          IF ABS(f.m[3]) > 0.1 GOTO 10
14          mass.before = f.m[3]
15      20  SIGNAL -1
16          RETURN
```

B.10 program OVERLOAD:

```
 1 ; program OVERLOAD (31.10.89 Kotzev)
 2 ;
 3 ; checking for overloads
 4          CALL jr3.dat
 5          FOR over = 1 TO 6
 6              fabs[over] = ABS(f.m[over])
 7              IF fabs[over] > 3*pcmax[over] THEN
 8                  SIGNAL 2030
 9              END
10          END
11          RETURN
```

B.11 program PCSTOP:

```
 1 ; program PCSTOP (Jan.12 /Kotzev)
 2 ;
 3          NOALTER
```

```
 4          SIGNAL -2020
 5          TIMER (1) = 0
 6          WAIT TIMER(1) > 1
 7          PCEND
 8          SIGNAL -2010, -2011, -2012, -2013, -2014, -2015, -2016
 9          SIGNAL -2017, -2018, -2019, 2020, -2025
10          RETURN
```

B.12 program REPEAT.BOLT:

```
 1 ; REPEAT.BOLT (June 18 - Kotzev)
 2 ;
 3          FOR repeat = 2 TO 4
 4              CALL screw1.ini1
 5              IF repeat == 2 THEN
 6                  SET tool.pnt = SHIFT(screw1.pnt1 BY -25, 0, 0)
 7              END
 8              IF repeat == 3 THEN
 9                  CALL screw1.ini1
10                  SET tool.pnt = SHIFT(screw1.pnt1 BY -50, 0, 0)
11              END
12              IF repeat == 4 THEN
13                  SET tool.pnt = SHIFT(screw1.pnt1 BY -75, 0, 0)
14              END
15              CALL pick
16              CALL screw1.ini2
17              IF repeat == 2 THEN
18                  SET tool.pnt = SHIFT(screw1.pnt2 BY 60, -60, 0)
19              END
20              IF repeat == 3 THEN
21                  SET tool.pnt = SHIFT(screw1.pnt2 BY 0, -60, 0)
22              END
23              IF repeat == 4 THEN
24                  SET tool.pnt = SHIFT(screw1.pnt2 BY 60, 0, 0)
25              END
26              CALL place
27          END
28          RETURN
```

B.13 program REPEAT.SCREW:

```
 1 ; REPEAT.SCREW (Sep. 10 - Kotzev)
 2 ;
 3          screw1 = 10
 4          screw2 = 10
```

```
 5          screw3 = 1
 6          screw4 = 1
 7          SIGNAL -2001
 8          FOR i = 1 TO 20
 9              com.z = 15*i; Movement due to screw pitch
10              FOR repeat = 1 TO 4
11                  FOR same.screw = 1 TO 3
12                      CALL tool.ini2
13                      IF repeat == 1 THEN
14                          IF screw1 == 10 GOTO 50
15                          SET tool.pnt = SHIFT(tool.pnt2 BY 0, 0, 0)
16                      END
17                      IF repeat == 2 THEN
18                          IF screw2 == 10 GOTO 50
19                          SET tool.pnt = SHIFT(tool.pnt2 BY 60, -60, 0)
20                      END
21                      IF repeat == 3 THEN
22                          IF screw3 == 10 GOTO 50
23                          SET tool.pnt = SHIFT(tool.pnt2 BY 0, -60, 0)
24                      END
25                      IF repeat == 4 THEN
26                          IF screw4 == 10 GOTO 50
27                          SET tool.pnt = SHIFT(tool.pnt2 BY 60, 0, 0)
28                      END
29                      CALL screw
30      50              IF SIG(2001) THEN
31                          IF repeat == 1 THEN
32                              screw1 = 10
33                          END
34                          IF repeat == 2 THEN
35                              screw2 = 10
36                          END
37                          IF repeat == 3 THEN
38                              screw3 = 10
39                          END
40                          IF repeat == 4 THEN
41                              screw4 = 10
42                          END
43                          SIGNAL -2001
44                      END
45                  END
46              END
47          END
48          RETURN
```

B.14 program ROTATE:

```
 1 ; subroutine ROTATE (June 21 - Kotzev)
 2 ;
 3         CALL pcstop
 4         TYPE "  ++++++++  ROTATE started "
 5         SPEED speed.rotate  ALWAYS
 6 ;       IF zero.position == 1 THEN
 7 ;       HERE #temp.pnt
 8 ;       DECOMPOSE a[] = #temp.pnt
 9 ;       MOVE #PPOINT(a[0], a[1], a[2], a[3], a[4], start[5])
10 ;       BREAK
11 ;       GOTO 100
12 ;       END
13         TYPE "    --------  ROTATE.PC started "
14 ;       PCEXECUTE rotate.pc, -1, 1
15 ;       ALTER (-1, 17, , 1)
16         FOR rotate = 1 TO rot.steps
17             HERE temp.pnt
18             DECOMPOSE a[] = temp.pnt
19             r[5] = a[5]+rot.dir*delta.o
20             SET temp.pnt = TRANS(a[0], a[1], a[2], a[3], a[4], r[5])
21             MOVES temp.pnt
22             BREAK
23             IF SIG(2015) GOTO 100
24         END
25    100  CALL pcstop
26         TYPE "  ++++++++  ROTATE ended "
27         RETURN
```

```
 1 ; ROTATE.PAR (June 21 - Kotzev)
 2 ;
 3         speed.rotate = 100
 4         rot.dir = -1
 5         delta.o = 10
 6         rot.steps = 30
 7         torque[5] = 300
 8         rotmax[3] = -0.7
 9         rotmin[3] = -1
10         del.rotmax[3] = 5
11         RETURN
```

```
 1 ; ROTATE.PC (June 19 - Kotzev)
```

```
 2 ;
 3 ; halting PC program when SIGNAL changes:
 4         IF SIG(2020) == 0 THEN
 5             TYPE "    --------  ROTATE.PC ended "
 6             HALT
 7         END
 8 ;
 9         CALL jr3.dat
10 ;force calculation and overload checking:
11         FOR pc = 1 TO 3
12             fabs[pc] = ABS(f.m[pc])
13             IF fabs[pc] > flimit[pc] THEN
14                 SIGNAL 2013
15             END
16         END
17 ; Check torques around Z axis:
18         IF ABS(f.m[6]) > torque[5] THEN
19             SIGNAL 2015
20         END
21 ; Monitor forces in the Z direction:
22         fsign[3] = 0
23         IF f.m[3] > rotmax[3] THEN; above the env.
24             fsign[3] = -ABS(f.m[3]-rotmax[3])*gain.in[3]
25         END
26         IF f.m[3] < rotmin[3] THEN
27             fsign[3] = ABS(f.m[3]-rotmin[3])*gain.out[3]
28         END
29         del[3] = fsign[3]
30         IF ABS(fsign[3]) > del.rotmax[3] THEN
31             IF fsign[3] == 0 GOTO 15
32             del[3] = del.rotmax[3]*fsign[3]/ABS(fsign[3])
33     15  END
34         ALTOUT 0, 0, 0, del[3], 0, 0, 0
```

B.15 SCREW parameters:

```
 1 ; program SCREW1.INI1 (June 6 - Kotzev)
 2 ;
 3         CALL pcstop
 4 ; CONFIGURATION:
 5         z.sensor = 31
 6         z.rcc = 43
 7         z.gripper = 112
 8         TOOL TRANS(0, 0, z.sensor+z.rcc+z.gripper, 90, -90, 45)
 9         mass = 0.17
```

```
10          ds[1] = 8
11          ds[2] = 8
12          z.tool = 40
13          d[3] = z.rcc+z.gripper+z.tool
14  ; MOVE parameters:
15          SET tool.pnt = screw1.pnt1
16          safe.z = 300
17          speed.move = 100
18  ; APPROACH parameters:
19          CALL approach.par
20  ; GRASP parameters:
21          temp.force = 10
22          speed.grasp = 10
23          weight.z = 50
24  ; CLEAR parameters:
25          CALL clear.par
26          RETURN
```

```
 1  ; SCREW1.INI2 (June 6 - Kotzev)
 2  ;
 3          CALL pcstop
 4  ; CONFIGURATION:
 5          z.tool = 40
 6          TOOL TRANS(0, 0, z.sensor+z.rcc+z.gripper+z.tool, 90, -90, 45)
 7          mass = 0.17
 8          ds[1] = 8
 9          ds[2] = 8
10          d[3] = z.rcc+z.gripper+z.tool
11  ; MOVE parameters:
12          SET tool.pnt = screw1.pnt2
13          safe.z = 300
14          speed.move = 100
15  ; APPROACH parameters:
16          CALL approach.par
17          weight.ex = 0
18          open.grip = 0
19          shift.z = 15
20          speed.appro = 20
21          delta.shift = 1
22  ; CLEAR parameters:
23          CALL clear.par
24          clear.rotate = 1
25          RETURN
```

B.16 program SEARCH:

```
 1 ; subroutine SEARCH (June 17 - Kotzev)
 2 ;
 3         CALL pcstop
 4         TYPE "  ++++++++  SEARCH started "
 5         TYPE "    --------  SEARCH.PC started "
 6         PCEXECUTE search.pc, -1, 1
 7         ALTER (-1, 19, , 1)
 8 ; If rotate==1, rotate the part:
 9         IF rotate == 1 THEN
10             FOR search = 1 TO 10
11                 r3 = r[3]+rot[3]
12                 r4 = r[4]+rot[4]
13                 r5 = r[5]+rot[5]
14                 SET temp.pnt = TRANS(r[0], r[1], r[2], r3, r4, r5)
15                 MOVES HERE:temp.pnt
16                 BREAK
17             END
18         END
19         SPEED speed.xy  ALWAYS
20 ; Reset parameters for first force signal:
21     60  cycle = 0
22         count.search = 0
23         nstep = 0
24 ; Check if movement exceeded delsearch.z:
25         HERE ini.pnt
26         DECOMPOSE b[] = ini.pnt
27     50  HERE temp.pnt
28         DECOMPOSE a[] = temp.pnt
29         IF ABS(b[2]-a[2]) > delsearch.z THEN
30             TYPE "-----  reached min. location "
31             GOTO 100
32         END
33 ; Movement parameters:
34         stepx = 0
35         stepy = 0
36      5  IF count.search == 0 THEN
37             cycle = cycle+1
38             nstep = nstep+1
39             count.search = 1
40             IF SIG(2010) GOTO 15
41             stepx = (ABS(b[0]-a[0])+delx*cycle)/nstep
42             IF SIG(2011) GOTO 20
43             GOTO 10
```

```
44          END
45      15  IF count.search == 1 THEN
46              count.search = 2
47              IF SIG(2011) GOTO 25
48              stepy = (ABS(b[1]-a[1])+dely*cycle)/nstep
49              IF SIG(2010) GOTO 30
50              GOTO 10
51          END
52      25  IF count.search == 2 THEN
53              nstep = nstep+1
54              count.search = 3
55              IF SIG(2010) GOTO 35
56              stepx = -(ABS(b[0]-a[0])+delx*cycle)/nstep
57              IF SIG(2011) GOTO 20
58              GOTO 10
59          END
60      35  IF count.search == 3 THEN
61              count.search = 0
62              IF SIG(2011) THEN
63                  IF SIG(2010) GOTO 100
64                  GOTO 5
65              END
66              stepy = -(ABS(b[1]-a[1])+dely*cycle)/nstep
67              IF SIG(2010) GOTO 30
68              GOTO 10
69          END
70      10  FOR search = 1 TO nstep
71              SET temp.pnt = SHIFT(HERE BY stepx, stepy, 0)
72              MOVES temp.pnt
73              BREAK
74              IF SIG(2010) GOTO 60
75              IF SIG(2011) GOTO 60
76          END
77          GOTO 50
78      20  FOR search = 1 TO nstep
79              SET temp.pnt = SHIFT(HERE BY stepx, 0, 0)
80              MOVES temp.pnt
81              BREAK
82              IF SIG(2010) GOTO 100
83          END
84          GOTO 50
85      30  FOR search = 1 TO nstep
86              SET temp.pnt = SHIFT(HERE BY 0, stepy, 0)
87              MOVES temp.pnt
88              BREAK
```

```
            IF SIG(2011) GOTO 100
        END
        GOTO 50
; Search movements ended, rotate the part:
 100  IF rotate == 1 THEN
            FOR search = 1 TO 10
                r3 = r[3]-rot[3]
                r4 = r[4]-rot[4]
                r5 = r[5]-rot[5]
                SET temp.pnt = TRANS(r[0], r[1], r[2], r3, r4, r5)
                MOVES HERE:temp.pnt
                BREAK
            END
        END
        CALL pcstop
        TYPE "  ++++++++  SEARCH finished "
        RETURN
```

```
; SEARCH.PAR (May 26 /Kotzev)
;
        rotate = 0;     ROTATE par.
        r[0] = 0
        r[1] = 0
        r[2] = 0
        r[3] = 0
        r[4] = -90
        r[5] = 90
        rot[3] = 0
        rot[4] = 0
        rot[5] = 0
;
        count.max = 50
        delx = 0.5
        dely = 0.5
        search.gain[3] = 10
        speed.xy = 5
        speed.rotate = .1E-01
        semax[1] = 1
        semin[1] = -1
        semax[2] = 1
        semin[2] = -1
        semax[3] = -0.7
        semin[3] = -1
```

```
26          del.semax[3] = 5
27          gain.in[3] = 1.5
28          gain.out[3] = 0.2
29          flimit[1] = 3
30          flimit[2] = 3
31          flimit[3] = 5
32          delsearch.z = 5
33          RETURN

 1 ; SEARCH.PC (June 6 - Kotzev)
 2 ;
 3 ; halting PC program when SIGNAL changes:
 4          IF SIG(2020) == 0 THEN
 5              TYPE "    --------  SEARCH.PC ended "
 6              HALT
 7          END
 8 ;
 9          CALL jr3.dat
10 ;force calculation and overload checking:
11          FOR pc = 1 TO 3
12              fabs[pc] = ABS(f.m[pc])
13              IF fabs[pc] > flimit[pc] THEN
14                  SIGNAL 2013
15              END
16          END
17 ; Check force in the X direction:
18          IF fabs[1] > semax[1] THEN
19              SIGNAL 2010
20          END
21 ; Check force in the Y direction:
22          IF fabs[2] > semax[2] THEN
23              SIGNAL 2011
24          END
25 ; Monitor forces in the Z direction:
26          fsign[3] = 0
27          IF f.m[3] > semax[3] THEN; above the env.
28              fsign[3] = -ABS(f.m[3]-semax[3])*gain.in[3]
29          END
30          IF f.m[3] < semin[3] THEN
31              fsign[3] = ABS(f.m[3]-semin[3])*gain.out[3]
32          END
33          del[3] = fsign[3]
34          IF ABS(fsign[3]) > del.semax[3] THEN
35              IF fsign[3] == 0 GOTO 15
36              del[3] = del.semax[3]*fsign[3]/ABS(fsign[3])
```

```
37      15  END
38          ALTOUT 0, 0, 0, del[3], 0, 0, 0
```

B.17 program SHOW.FM:

```
 1 ; program SHOW.FM (Feb. 26. 90 Kotzev)
 2 ;
 3 ;
 4          DO
 5              CALL jr3.dat
 6              TYPE "Fx [Kg]   =", /F6.2, f.m[1], "    Fy [Kg]   =",
                   /F6.2, f.m[2], "    Fz [Kg]   =", /F6.2, f.m[3]
 7              TYPE "Mx [Kg*mm]=", /F7.2, f.m[4], "   My [Kg*mm]=",
                   /F7.2, f.m[5], "   Mz [Kg*mm]=", /F7.2, f.m[6]
 8              weight = SQRT(f.m[1]*f.m[1]+f.m[2]*f.m[2]+f.m[3]*f.m[3])
 9              TYPE "Weight= ", weight
10              TYPE "press  REC  botton on teach pendant to stop"
11              TYPE "  "
12              TIMER (1) = 0
13              WAIT TIMER(1) > 5
14          UNTIL (PENDANT(1) BAND 1) <> 0
```

B.18 program TIP:

```
 1 ; program TIP (Jan.10 /Kotzev)
 2 ;
 3          CALL jr3.dat
 4          fmax[4] = 75
 5 ;
 6 ;
 7          f[1] = z.d1*.122E-01
 8          f[2] = z.d2*.122E-01
 9          f[3] = z.d3*.244E-01
10          f[4] = z.d4*.336E-01
11          f[5] = z.d5*.336E-01
12          f[6] = z.d6*0
13 ;
14 ;        FOR jr = 1 TO 6
15 ;        IF ABS(f[jr]) > fmax[jr] THEN
16 ;        TYPE "JR3 error - OVERLOAD"
17 ;        END
18 ;        END
19 ;
20          f.m[1] = 0.445*f[1]
21          f.m[2] = 0.445*f[2]
```

```
22          f.m[3] = -0.445*f[3]
23          f.m[4] = 11*f[4]-ds[3]*f.m[2]+ds[2]*f.m[3]
24          f.m[5] = 11.3*f[5]+ds[3]*f.m[1]-ds[1]*f.m[3]
25          f.m[6] = -11.3*f[6]-ds[2]*f.m[1]+ds[1]*f.m[2]
26          HERE point
27          DECOMPOSE a[] = point
28          nx = -SIN(a[3])*SIN(a[4])*COS(a[5])+COS(a[3])*SIN(a[5])
29          ny = COS(a[3])*SIN(a[4])*COS(a[5])+SIN(a[3])*SIN(a[5])
30          nz = -COS(a[4])*COS(a[5])
31          sx = SIN(a[3])*SIN(a[4])*SIN(a[5])+COS(a[3])*COS(a[5])
32          sy = -COS(a[3])*SIN(a[4])*SIN(a[5])+SIN(a[3])*COS(a[5])
33          sz = COS(a[4])*SIN(a[5])
34          ax = SIN(a[3])*COS(a[4])
35          ay = -COS(a[3])*COS(a[4])
36          az = -SIN(a[4])
37          f[1] = nx*f.m[1]+sx*f.m[2]+ax*f.m[3]
38          f[2] = ny*f.m[1]+sy*f.m[2]+ay*f.m[3]
39          f[3] = nz*f.m[1]+sz*f.m[2]+az*f.m[3]
40          f[4] = f.m[4]
41          f[5] = f.m[5]
42          f[6] = f.m[6]
43     500  RETURN
```

B.19 TOOL parameters:

```
 1 ; program TOOL.INI1 (June 17 - Kotzev)
 2 ;
 3          CALL pcstop
 4 ; CONFIGURATION:
 5          z.sensor = 31
 6          z.rcc = 43
 7          z.gripper = 112
 8          TOOL TRANS(0, 0, z.sensor+z.rcc+z.gripper, 90, -90, 45)
 9          mass = 0.17
10          ds[1] = 8
11          ds[2] = 8
12          z.tool = 40
13          ds[3] = z.rcc+z.gripper+z.tool
14 ; MOVE parameters:
15          SET tool.pnt = tool.pnt1
16          safe.z = 300
17          speed.move = 100
18 ; APPROACH parameters:
19          CALL approach.par
20 ; GRASP parameters:
```

```
21          temp.force = 10
22          speed.grasp = 10
23          weight.z = 50
24  ; CLEAR parameters:
25          CALL clear.par
26          clear.rotate = 1
27          clear[5] = -260
28          RETURN
```

```
 1  ; TOOL.INI2 (June 21 - Kotzev)
 2  ;
 3          CALL pcstop
 4  ; CONFIGURATION:
 5          z.tool = 31
 6          TOOL TRANS(0, 0, z.sensor+z.rcc+z.gripper+z.tool, 90, -90, 45)
 7          mass = 0.17
 8          ds[1] = 10
 9          ds[2] = 10
10          d[3] = z.rcc+z.gripper+z.tool
11  ; MOVE parameters:
12          SET tool.pnt = tool.pnt2
13          safe.z = 25
14          speed.move = 100
15  ; APPROACH parameters:
16          CALL approach.par
17          speed.high = 5
18          speed.appro = 5
19          weight.ex = 0
20          open.grip = 0
21          rot.appro = 1
22          shift.z = 4
23          shift.over = 15
24          speed.appro = 10
25          delta.shift = 1
26  ; COMSCREW pararmeters:
27          CALL comscrew.par
28          lower.pnt = 20
29  ; CLEAR parameters:
30          CALL clear.par
31          speed.clear = 20
32          clear.rotate = 1
33          speed.rotat = 100
34          RETURN
```

B.20 TOP parameters:

```
 1 ; TOP.INI1 (June 5 - Kotzev)
 2 ;
 3         CALL pcstop
 4         TYPE "  "
 5         TYPE " TOP ASSEMBLY "
 6         TYPE "  "
 7 ; CONFIGURATION:
 8         z.sensor = 31
 9         z.rcc = 43
10         z.gripper = 112
11         TOOL TRANS(0, 0, z.sensor+z.rcc+z.gripper, 90, -90, 45)
12         mass = 0.17
13         ds[1] = 8
14         ds[2] = 8
15         z.tool = 20
16         d[3] = z.rcc+z.gripper+z.tool
17 ; MOVE parameters:
18         SET tool.pnt = top.pnt1
19         safe.z = 300
20         speed.move = 100
21 ; APPROACH parameters:
22         CALL approach.par
23         shift.x = 25
24         delta.y = 0
25         delta.z = 0
26 ; GRASP parameters:
27         temp.force = 1
28         speed.grasp = 10
29         weight.z = 30
30 ; CLEAR parameters:
31         CALL clear.par
```

```
 1 ; TOP.INI2 (June 5 - Kotzev)
 2 ;
 3         CALL pcstop
 4 ; CONFIGURATION:
 5         z.tool = 0; rotation axis
 6         TOOL TRANS(0, 0, z.sensor+z.rcc+z.gripper+z.tool, 90, -90, 45)
 7         mass = 0.5
 8         ds[1] = 80
 9         ds[2] = 20
10         d[3] = z.rcc+z.gripper+z.tool
```

```
11 ; MOVE parameters:
12         SET tool.pnt = top.pnt2
13         safe.z = 300
14         speed.move = 100
15 ; APPROACH parameters:
16         CALL approach.par
17         weight.ex = 0
18         open.grip = 0
19         shift.z = 0
20         shift.over = 0
21         position.appro = 0
22         appro.max[3] = 0.2 ; no contact during approach
23 ; SEARCH parameters:
24         CALL search.par
25         rotate = 1
26         rot[4] = 2.5; multiply 10 times
27         pc[1] = 0.5
28         pc[2] = 0.5
29         pc[3] = 0.5
30         delsearch.z = 10
31         gain.in[3] = 0.3
32         gain.out[3] = 0.1
33         delx = 1
34         dely = 1
35 ; COMPLY parameters:
36         CALL comply.par
37         lower.pnt = 25
38         speed.comply = 4
39         comax[3] = -1.5
40         comin[3] = -1.4
41         comin[5] = -250
42         comaz[5] = -200
43         gain[4] = 0
44         gain[6] = 0
45 ; CLEAR parameters:
46         CALL clear.par
47         clear.grip = 1
48         clear.y = 15
```

B.21 program WEIGHT:

```
 1 ; WEIGHT (June 16 - Kotzev)
 2 ;
 3         PCEND
 4         sigma1 = 0
```

```
 5          samples = 25
 6          FOR i = 1 TO samples
 7             [i] = f.m[3]
 9              sigma1 = sigma1+force[i]
10          END
11          sigaverage = sigma1/samples
12          sigdel = ABS(sigaverage*0.1)
13          sigma2 = 0
14          tempcount = 0
15          FOR i = 1 TO samples
16              IF ABS(ABS(force[i])-ABS(sigaverage)) > sigdel THEN
17                  sigma2 = sigma2+force[i]
18                  tempcount = tempcount+1
19              END
20          END
21          IF tempcount > 0 THEN
22              weight = sigma2/tempcount
23          ELSE
24              weight = sigaverage
25          END
26          weight = ABS(weight)
27          RETURN
```