

Section II

Robotics

Revised 8/2/05

Table of Contents

1.0	<i>General Lab (AML) Objectives</i>	5
2.0	<i>Student Responsibilities</i>	5
3.0	<i>General Specifications of Lab Equipment</i>	6
4.0	<i>Robotics Safety</i>	9
5.0	<i>Useful Robot Programming/Teaching Techniques</i>	10
	<i>Introductory Lecture to Robotics: Prof. Steven Derby</i>	13
1.	<i>Introduction</i>	13
1.1	<i>Automation and Robotics</i>	13
1.2	<i>Robotics Market and Future Prospects</i>	14
2.	<i>Fundamentals of Robot Technology, Programming, and Applications</i>	15
2.1	<i>Robot Anatomy</i>	15
2.2	<i>Work Volume</i>	17
2.3	<i>Robot Drive Systems</i>	17
2.4	<i>Control Systems and Dynamic Performance</i>	19
2.5	<i>Precision of Movement</i>	20
2.6	<i>End Effectors</i>	22
2.7	<i>Robotic Sensors</i>	22
2.8	<i>Robotic Programming and Work Cell Control</i>	23
2.9	<i>Robot Applications</i>	23
2.10	<i>Jacobians</i>	24

Appendices

A.	<i>V+ Commands</i>	26
----	--------------------------	----

List of Figures

2.1	<i>Staubli RX90 Work Area</i>	7
2.2	<i>Staubli Joint Orientation</i>	8
2.3	<i>Adept Work Area</i>	8
2.4	<i>Adept Joint Orientation</i>	9
2.5	<i>Classes of Industrial Automation</i>	13
2.6	<i>Actual and Projected Industrial Robot Sales</i>	14
2.7	<i>Robot Configurations</i>	15
2.8	<i>Joint Motions</i>	16
2.9	<i>Degrees of Freedom in Wrist</i>	17
2.10	<i>Robot Anatomies</i>	17
2.11	<i>Hydraulic and Electric Drives</i>	18
2.12	<i>Influence of Distance vs. Speed</i>	19
2.13	<i>Damping vs. Response</i>	20
2.14	<i>Accuracy</i>	21
2.15	<i>Resolution</i>	21
2.16	<i>Illustration of Repeatability and Accuracy</i>	22

1.0 GENERAL LAB (AML) OBJECTIVES

The main objective of AML is to teach students to combine analytical thinking, basic knowledge, and common sense to solve practical engineering problems. We therefore ask you to think about what you are doing at all times from a safety and time utility point of view. Don't be afraid to THINK. There won't always be an answer key. As a result of these lab exercises, you should gain basic knowledge about automation, the importance of degrees of freedom, rigidity, repeatability, feedback and control in robot design, and computer programming (don't let that scare you, it will be fun). In the process you will gain confidence in your abilities which you will take with you wherever you go. The labs are not designed to make you experts in the robot operating systems you will be using, but you should walk away with a feel for automation's problems, limitations, and applications as well as discovering how much fun it can be to teach something (the robot) with an IQ of zero. (More on this later).

The lab equipment consists of two robotics systems, both which have input/output capability. In other words, the operating systems can be used as over glorified programmable logic controllers. The first set up consists of a Staubli Six Degree of Freedom (DOF) robot and a Sony Conveyor System. The second setup consists of an Adept Four DOF robot. You will be using both systems to perform basic automation tasks.

2.0 STUDENT RESPONSIBILITIES

Each lab section will be divided into two groups of approximately 3 people each. Please divide yourself into these groups before the first lab period and decide which group will be group A and which will be group B. Group A will start on the Staubli and group B will start on the Adept.

Your first responsibility is almost fulfilled. Before coming to lab the first day, read up to and including the section entitled useful robot programming techniques plus the sheet on the actual lab you will be doing. Looking over the program and the commands descriptions in the appendix will give you a good start.

3.0 GENERAL SPECIFICATIONS OF LAB EQUIPMENT

	<u>STAUBLI RX90</u>	<u>ADEPT COBRA 800</u>
Axes	6	4
Drive mechanism	DC servo motors/gears	DC servo motors/belts
System language	V+	V+
Programming unit	Terminal	Terminal
Program storage	Floppy disk / Hard Drive	Ethernet / Hard Drive
I/O Capability	32 outputs 48 digital inputs	40 outputs 44 digital inputs
Arm weight	250 lb.	77 lb.
Max load on arm (inc. gripper)	15 lb.	12.1 lb.
Repeatability	0.0008"	0.0008"
Max velocity	60 ips	40 ips
Max acceleration	1 g	0.75g
Max input air pressure	90 psi	90 psi

Sony Conveyor

Configuration	T or in line, controllable
# of pallets	Can track 3 simultaneously
Control	Internal pallet tracking by programmable logic controller
Number of workstations	Five which can be positioned on any straight section All five can be controlled manually Two are wired to Staubli controller for automatic control Two are wired to Staubli controller for automatic control
Repeatability of pallet locating	0.005"
Speed	~ 1 in/sec

*See Sony blueprints for detailed geometric information

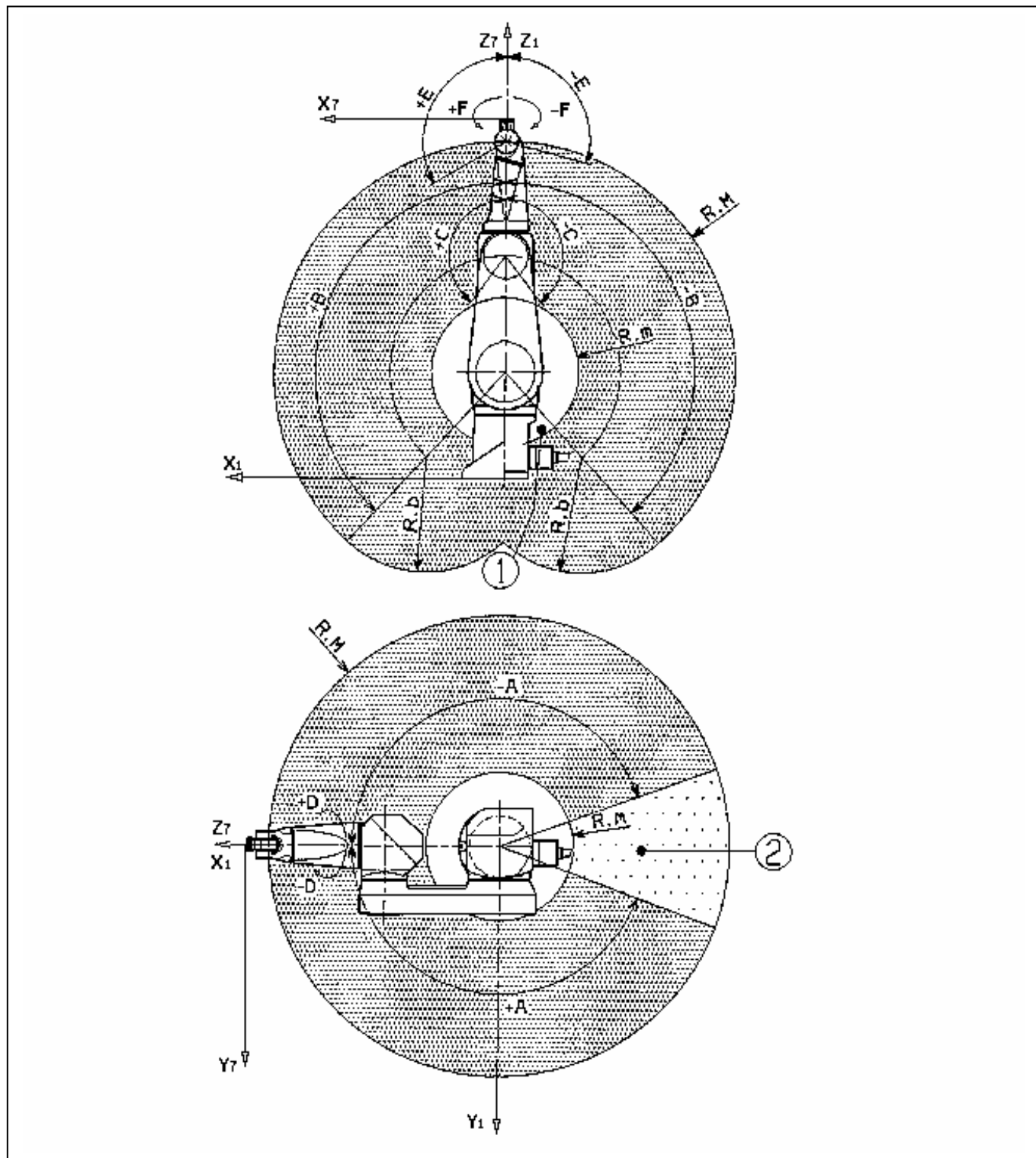


Figure 2.1: STAUBLI RX90 Work Area

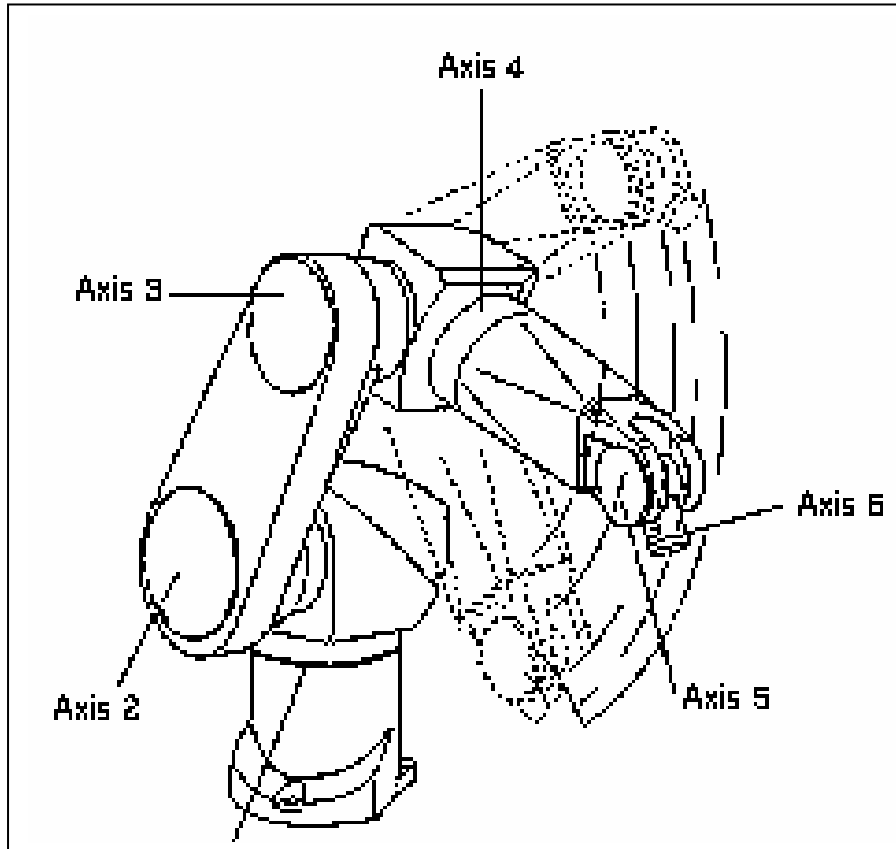


Figure 2.2: Staubli Joint Orientation

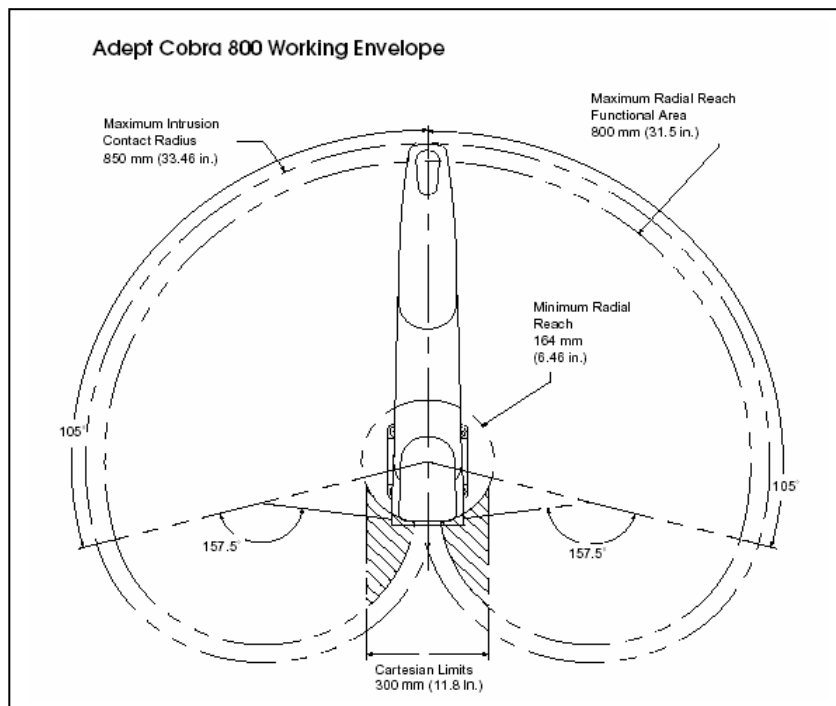


Figure 2.3: Adept Work Area

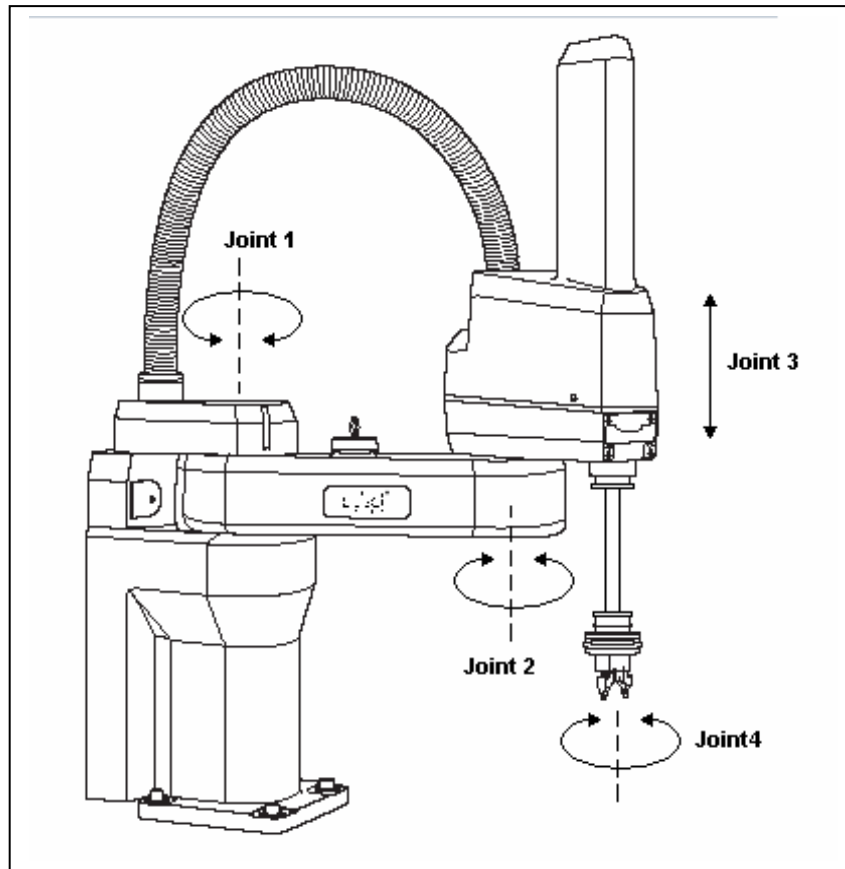


Figure 2.4: Adept Joint Orientation

4.0 ROBOTIC SAFETY

The equipment you are using is not a toy. It is actual industrial equipment and can cause serious injury. In motion, these robots have a lot of inertia and can behave unpredictably. In addition to dangers of being hit by the robot, there are dangers of parts flying off the robot arm and dangers of electric shock. To maintain maximum safety during robot operation, the following guidelines are given.

The Ten Commandments of Robotics Lab Safety

1) TO STOP ROBOT MOTION

STAUBLI

- 1) RED MUSHROOM BUTTON ON TEACH PENDANT**
- 2) ARM POWER OFF MUSHROOM ON CONTROLLER**
- 3) DEAD MAN'S SWITCH ON TEACH PENDANT**

ADEPT SCARA

- 1) RED MUSHROOM BUTTON ON TEACH PENDANT**
- 2) MUSHROOM SWITCH ON CONTROLLER INTERFACE PANEL**
- 3) DEAD MAN'S SWITCH ON TEACH PENDANT**

- 2) ALWAYS KEEP WORK ENVELOPE CLEAR OF PEOPLE**
- 3) DO NOT OPERATE ROBOT IF YOU SEE LOOSE CONNECTIONS OR HEAR PNEUMATIC LEAKS**
- 4) MOVE SLOWLY WHEN TEACHING POINTS**
- 5) RUN PROGRAM SLOWLY FOR FIRST EXECUTION (PREFERABLY IN STEP MODE) AND NEVER RUN A PROGRAM WITHOUT TA APPROVAL FIRST**
- 6) DO NOT ENTER NONSENSE WORDS INTO THE COMPUTER**
- 7) DO NOT TRY TO MOVE END EFFECTOR OUT OF ENVELOPE OR THROUGH ROBOT BASE**
- 8) ALWAYS HAVE TERMINAL ON WHEN USING ROBOT**
- 9) NEVER POWER DOWN A ROBOT AT A JOINT LIMIT STOP**

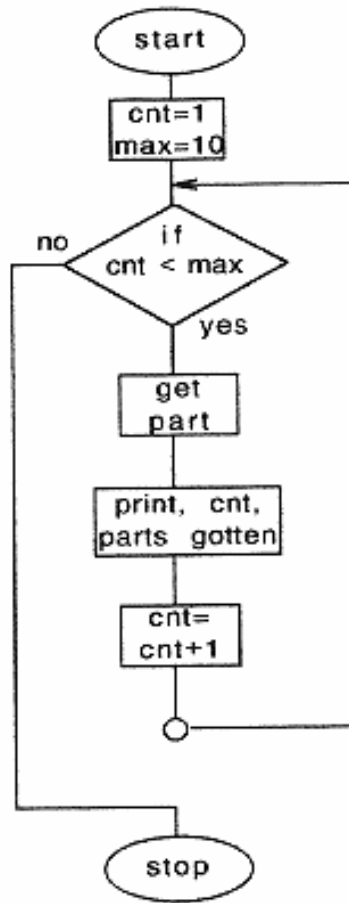
5.0 USEFUL ROBOT PROGRAMMING/TEACHING TECHNIQUES

When performing the lab exercises, follow these general operating guidelines:

- 1) The first step in writing a robotics program is to make a flow chart of the required tasks showing their order, number of repetitions, where the robot controller has to interact**

with its environment (I/O signals) etc.

- 2) **Always write the program before teaching points.** Otherwise you may forget to add the move statement into the program resulting in disaster. Also having a general idea of how you want the gripper to move helps you to teach points right the first time.
- 3) Use Delay type statements
 - a) after opening and closing finger grippers
 - b) after an air related output command (solenoid valve, cylinder, suction cup)
 - c) before looking for inputs from vacuum sensors
 - d) after a vacuum release has been sensed in vacuum grippers
 - e) whenever position accuracy is important to allow elastic vibrations to dissipate
- 4) Lower motion speed when approaching feeders and other tight spots
- 5) When teaching pallets, teach the relevant part of the pallet **WITHOUT** removing part from the gripper. Using pendant, position gripper with part in order to find new teach points.
- 6) (Staubli) Move robot in world coordinates for gross motions, tool coordinates for final alignments and approaches, and joint coordinates to get out of situations where limit stops are encountered.
- 7) Define end effector paths with multiple points to ensure that robot arm does not hit obstacles or try to go through its own base.
- 8) Perform all positioning shifts during program execution on temporary variables. Do not shift the position of a taught position in program. Assign a temporary variable the taught position when initializing the program and perform the shifts on that variable.
- 9) All programs have three basic parts:
 - I) The initialization where you ensure the robot is starting in the correct location, all variables and counters are defined, and all outputs are initialized. **ALWAYS START A PROGRAM BY HAVING THE ROBOT MOVE TO A SAFE POSITION. USUALLY THIS SAFE POSITION IS ONE WHERE THE ROBOT CAN MOVE TO/FROM ALL LOCATIONS THAT IT MAY GO TO DURING PROGRAM EXECUTION, WITHOUT HITTING ANY OBSTACLES.**
 - II) The part of the program where you tell the robot to move
 - III) The part of the program where you return robot to initial configuration so you know that program execution has terminated
- 10) Often robots are used to perform a number of repetitive tasks. An important control structure for repetitive operations is the **DO WHILE LOOP**:



V+ has a number of flow control methods to set up these loops: DO UNTIL, IF THEN, CASE OF ..., GOTO, etc...

INTRODUCTORY LECTURE TO ROBOTICS
PROFESSOR S. DERBY, FALL 1995

1.0 Introduction

Robotics is a prominent component of manufacturing automation which will affect human labor at all levels, from unskilled workers to professional engineers and managers of production. Robots are highly automated mechanical manipulators controlled by computers.

1.1 Automation and Robotics

Automation and robotics are two closely related technologies.

Automation is a technology that is concerned with the use of mechanical, electronic, and computer-based systems in the operation and control of production, e.g. mechanical assembly machines, feedback control systems applied to industrial processes, numerically controlled machine tools, and robots.

Robotics is a form of industrial automation. There are 3 broad classes of industrial automation:

- a) Fixed Automation is used when the volume of production is very high, and it is therefore appropriate to design specialized equipment to process the product (or a component of a product) very efficiently and at high production rates.
- b) Programmable Automation is used when the volume of production is relatively low, and there are a variety of products to be made. The production equipment is designed to be adaptable to variation in product configuration.
- c) Flexible Automation possesses some of the features of both fixed and programmable automation. It must be programmed for different product configurations, but the variety of configurations is usually more limited than for programmable automation.

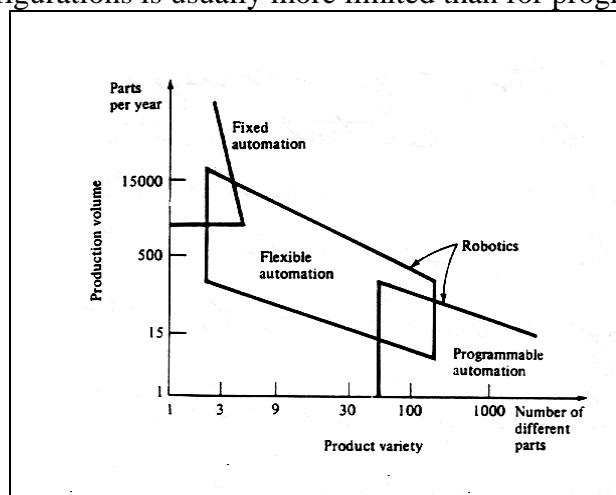
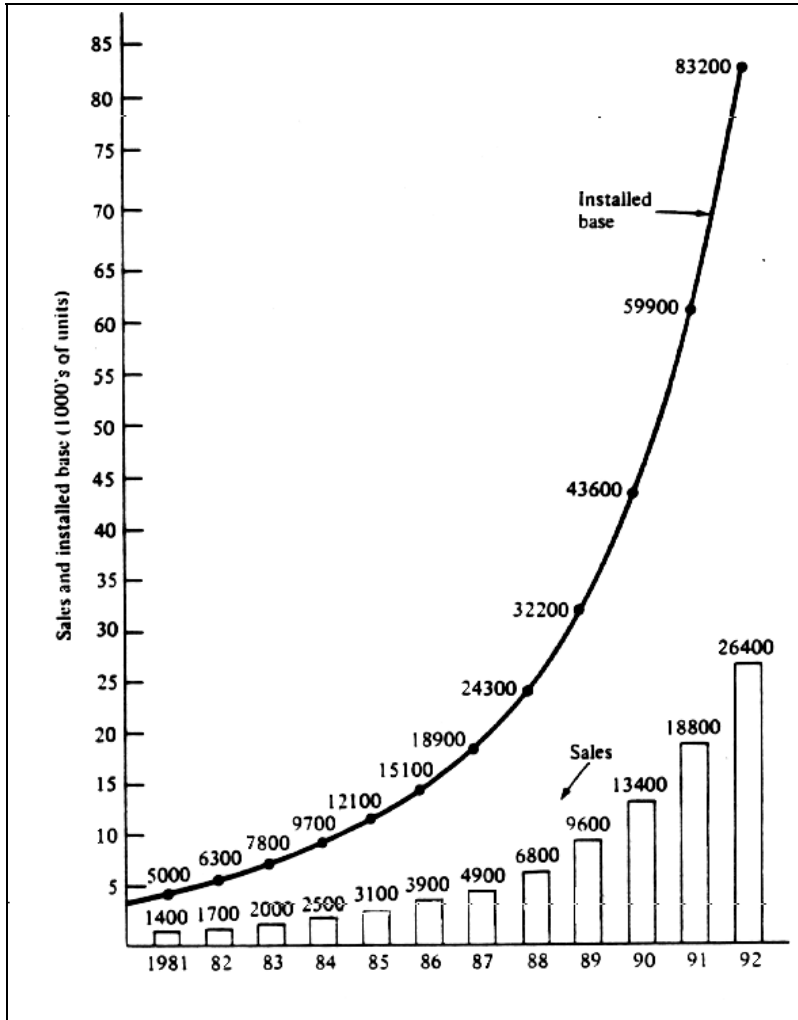


Figure 2.5: Classes of Industrial Automation

Of the three types of automation, robotics coincides most closely with programmable automation. An industrial robot is a reprogrammable, multifunctional manipulator designed to move materials, parts, tools, or special devices through variable programmed motions for the performance of a variety of tasks. While the robots themselves are examples of programmable automation, they are sometimes used in flexible automation or even fixed automation systems. Future robots are likely to have greater sensor capabilities, more intelligence, a higher level of manual dexterity, and a limited degree of mobility.

1.2 Robotics Market and Future Prospects



This figure shows actual and projected annual sales and the resulting number of installations of industrial robots (adjusted for obsolete robots that have been discarded) in the United States.

Figure 2.6: Actual and Projected Industrial Robot Sales

2.0 FUNDAMENTALS OF ROBOT TECHNOLOGY, PROGRAMMING, AND APPLICATIONS

Robotics is an applied engineering science that includes machine design, control theory, microelectronics, computer programming, artificial intelligence, human factors, and production theory.

2.1 Robot Anatomy

Robot Anatomy is concerned with the physical construction of the body, arm, and wrist of the machine. Most robots used in plants are mounted on a base which is fastened to the floor. The body is attached to the base, and the arm assembly is attached to the body. At the end of the arm is the wrist. The wrist consists of a number of components that allow it to be oriented in a variety of positions. Relative movements between the various components of the body, arm, and wrist are provided by a series of joints (usually either revolute or translational). The body, arm, and wrist assembly is called the manipulator. Attached to the robot's wrist is the end effector, which is not considered as part of the robot's anatomy. The arm and body joints of the manipulator are used to position the end effector, and the wrist joints of the manipulator are used to orient the end effector.

2.1.1 Common Robot Configurations

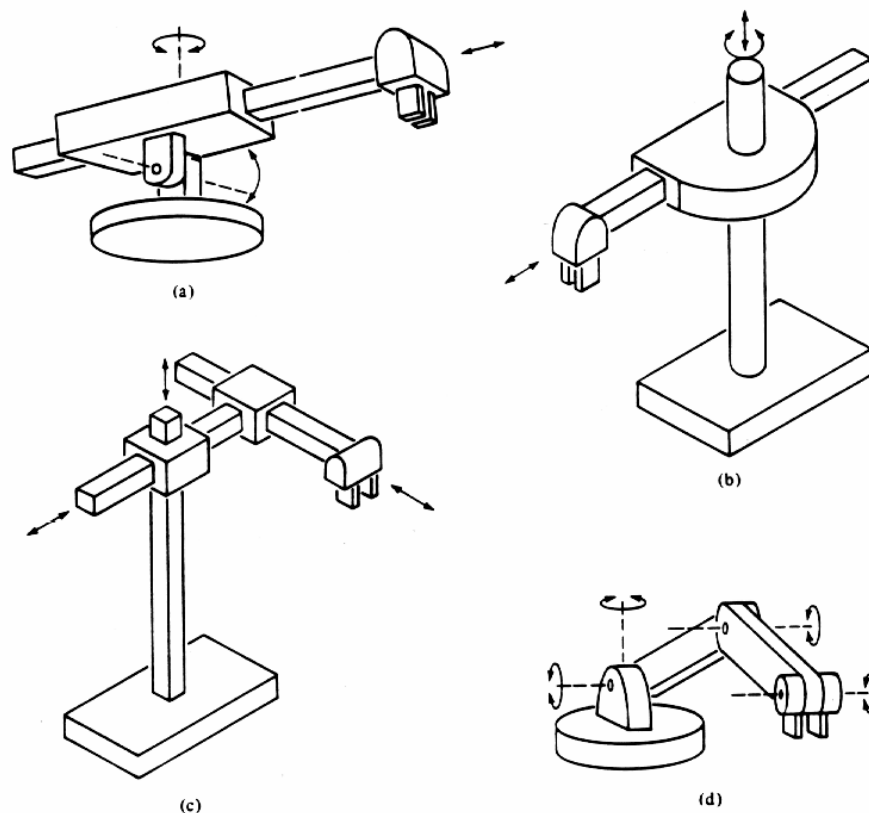


Figure 2.7: Robot Configurations

- a) Polar or spherical robot: The various joints provide the robot with the capability to move its arm within a spherical space.
- b) Cylindrical robot: This robot is capable of achieving a work space that approximates a cylinder.
- c) Cartesian, xyz, or rectilinear robot: This robot is capable of operating within a rectangular work envelope.
- d) Jointed arm robot: Its configuration is similar to that of a human arm.

There are relative advantages and disadvantages to the 4 basic robot anatomies simply because of their geometries. For example:

Repeatability of motion: The Cartesian robot probably possesses the advantage because of its inherently rigid structure.

Reach: The polar and jointed-arm robots have the advantage.

Lift capacity: Cylindrical and Cartesian robots can be designed for high-rigidity and load carrying capacity

Reach into a small opening: Polar and cylindrical robots possess a geometric advantage

2.1.2 Robot Motions

A robot's movements can be divided into two general categories:

- a) arm and body motions and
- b) wrist motions.

The individual joint motions associated with these two categories are sometimes referred to as degrees of freedom. A typical industrial robot is equipped with 4 to 6 degrees of freedom. The robot's motions are accomplished by means of powered joints. Connecting the various manipulator joints together are rigid members that are called links.

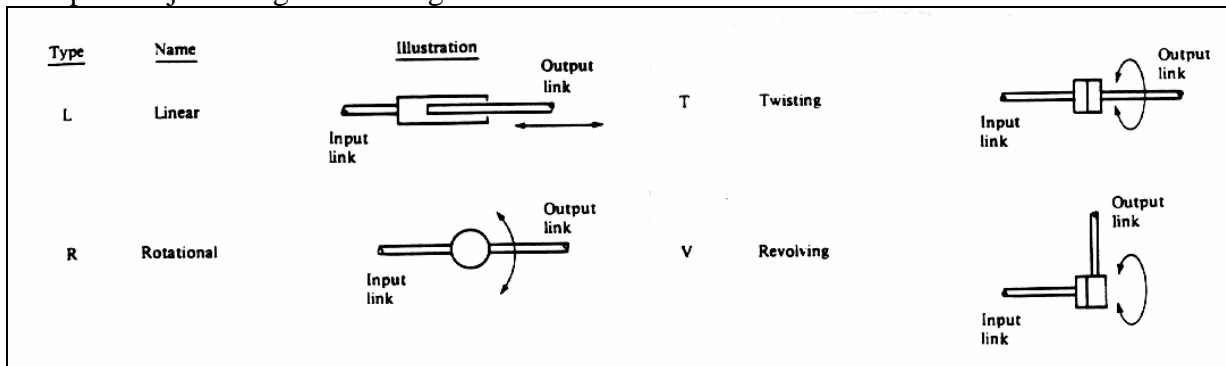


Figure 2.8: Joint Motions

The joints used in the design of industrial robots typically involve a relative motion of the adjoining links that is either linear or rotational.

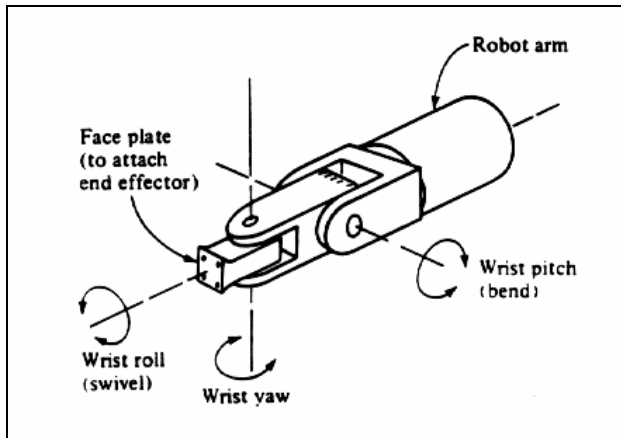


Figure 2.9: Degrees of freedom in wrist

The arm and body joints are designed to enable the robot to move its end effector to a desired position within the limits of the robot's size and joint movements. The wrist movement is designed to enable the robot to orient the end effector properly with respect to the task to be performed. To solve this orientation problem, the wrist is normally provided with up to 3 degrees of freedom, e.g. wrist roll, wrist pitch, and wrist yaw.

2.2 Work Volume

Work Volume refers to the space within which the robot can manipulate its wrist end. The convention of using the wrist end to define the robot's work volume is adopted to avoid the complication of different sizes of end effectors that might be attached to the robot's wrist.

The work volume is determined by the following physical characteristics of the robot:

- 1) the robot's physical configuration
- 2) the sizes of the body, arm, and wrist components
- 3) the limits of the robot's joint movements

Work Volumes for Various Robot Anatomies:

- a) *Polar*
- b) *Cylindrical*
- c) *Cartesian*

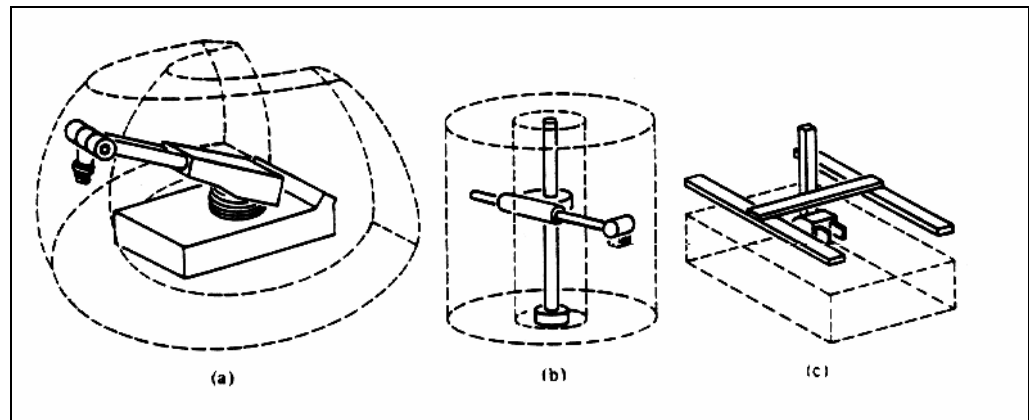


Figure 2.10: Robot Anatomies

2.3 Robot Drive Systems

The robot's capacity to move its body, arm, and wrist is provided by the drive system used to power the robot. The drive system determines the speed of the arm movements, the strength of the robot, its dynamic performance, and, to some extent, the kinds of applications that the robot can accomplish.

2.3.1 Types of Drive Systems

Commercially available industrial robots are powered by one of 3 types of drive systems:

- 1) Hydraulic
- 2) Electric
- 3) Pneumatic

1) Hydraulic Drive

- generally associated with larger robots
- provide robot with greater speed and strength
- adds to floor space required by robot
- inclined to leak oil
- can be designed to actuate either rotational or linear joints

2) Electric Drive

- do not provide as much speed or power as hydraulic systems
- tend to be smaller, requiring less floor space
- applications tend toward more precise work
- can be used to actuate linear or rotational joints
- cost of an electric motor is much more proportional to its size than a hydraulic drive system

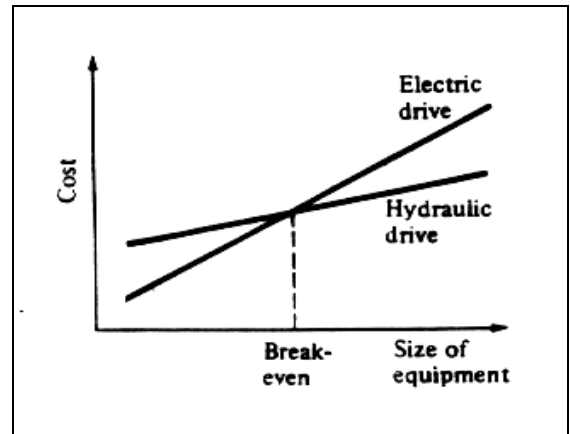


Figure 2.11 Hydraulic and Electric Drives

There is a trend in the design of industrial robots toward all electric drives and away from hydraulic robots.

3) Pneumatic Drive

- generally reserved for smaller robots that possess fewer degrees of freedom
- often limited to simple “pick-and-place” operations with fast cycles
- can be used to actuate linear and rotational joints

2.3.2 Speed of Motion

Speed determines how quickly the robot can accomplish a given work cycle. Determination of the most desirable speed, in addition to merely attempting to minimize the production cycle time, would also depend on:

- a) the accuracy with which the end effector must be positioned
- b) the weight of the object being manipulated
- c) the distances to be moved

There is generally an inverse relationship between the accuracy and the speed of robot motions. Heavier objects mean greater inertia and momentum, and the robot must be operated more

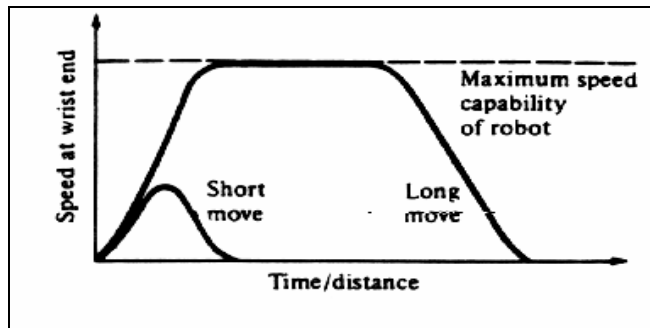


Figure 2.12: Influence of Distance vs. Speed

slowly to safely deal with these factors. The influence of distance of to be moved by the robot manipulator on speed of motion can be seen in the graph below.

2.3.3 Load-Carrying Capacity

The size, configuration, construction, and drive systems determine the load-carrying capacity of the robot. This load capacity should be specified under the condition that the robot's arm is in its weakest position. The manufacturer's specification of this feature is the gross weight capacity, and the user must consider the weight of the end effector.

2.4 Control Systems and Dynamic Performance

Commercially available industrial robots can be classified into 4 categories according to their control systems:

- 1) Limited-Sequence Robots (lowest level of control)
- 2) Playback Robots with Point-to-Point Control
- 3) Playback Robots with Continuous Path Control
- 4) Intelligent Robots (most sophisticated level of control)

Limited Sequence Robots

- Do not use servo control to indicate relative positions of the joints.
- Controlled by setting limit switches and/or mechanical stops to establish the endpoints of travel for each of their joints.
- Any of the three drive systems can be used with this type of control system; however, pneumatic drive is most common.
- Applications involve simple motions e.g. pick-and-place operations

Playback Robots

- A series of positions or motions are "taught" to the robot, recorded into memory, and then repeated by the robot under its own control.
- Usually have some form of servo-control to ensure that positions achieved by the robot are the positions that have been taught

- a) point-to-point robots
 - capable of performing motion cycles that consist of a series of desired point locations and related actions
 - they do not control the path taken by the robot to get from one point to the next
- b) continuous-path robots
 - capable of performing motion cycles in which the path followed by the robot is controlled
 - straight line motion is a common form of continuous-path control for industrial robots

Intelligent Robots

- possess the capability not only to play back a programmed motion cycle but to also interact with its environment in a way that seems intelligent
- can alter their programmed cycle in response to conditions in workplace
- can make logical decisions based on sensor data received from the operation
- have the capacity to communicate during the work cycle with humans or computer-based systems

2.4.1 Speed of Response and Stability

There are 2 important characteristics of dynamic performance related to control systems design.

Speed of Response - refers to the capability of the robot to move to the next position in a short amount of time. It is related to the robot’s motion speed and is also a function of the control system.

Stability - measure of the oscillations which occur in the arm during movement from one position to the next

It is desirable in control systems design to have good stability and fast response time. Unfortunately, these are competing objectives

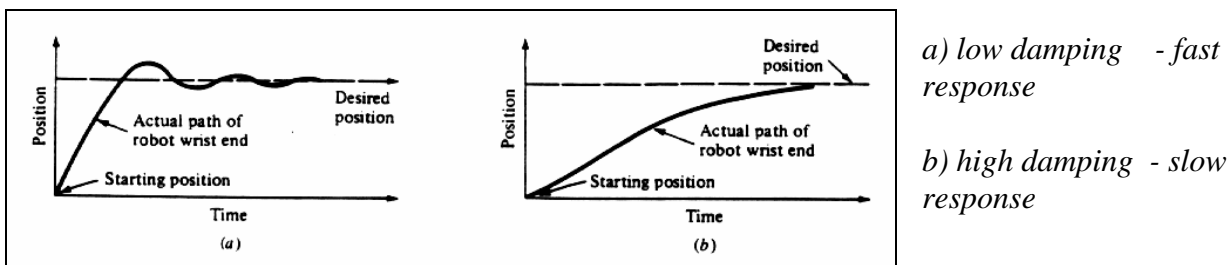


Figure 2.13: Damping vs. Response

2.5 Precision of Movement

Another measure of performance is precision of the robot’s movement. This is a function of:

- 1) Spatial Resolution
- 2) Accuracy
- 3) Repeatability

We will define these with the following assumptions:

- definitions will apply at robot's wrist end
- definitions apply to conditions under which the robot's precision will be at its worst
- definitions will be developed in the context of a point-to-point robot

2.5.1 Spatial Resolution

This is the smallest increment of movement into which the robot can divide its work volume. This depends on two factors: the system's control resolution and the robot's mechanical inaccuracies.

Control resolution is determined by the robot's position control system and its feedback measurement system.

Mechanical inaccuracies in the robot's links and joint components and its feedback measurement system come from elastic deflection in the structural members, gear backlash, stretching of pulley cords, leakage of hydraulic fluids, and other imperfections in the mechanical system. These inaccuracies are also influenced by such factors as the load being handled, the speed with which the arm is moving, etc.

The spatial resolution of the robot is the control resolution degraded by these mechanical inaccuracies

2.5.2 Accuracy

Accuracy refers to a robot's ability to position its wrist end at a desired target point within the work volume.

Figure 2.14 is an illustration of accuracy and control resolution when mechanical inaccuracies are assumed to be zero. This is the worst case assumption.

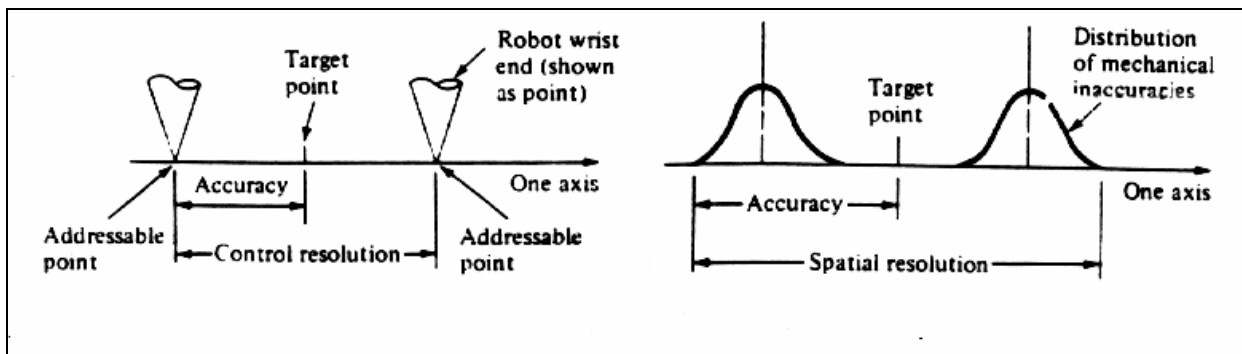


Figure 2.14: Accuracy

Figure 2.15: Resolution

Figure 2.15 displays the accuracy and spatial resolution in which mechanical inaccuracies are represented by a statistical distribution.

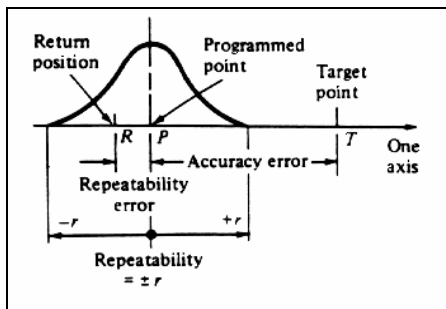
We define the robot's accuracy to be 1/2 of its spatial resolution.

Accuracy is affected by several factors:

- it varies within the work volume
- it is improved if the motion cycle is restricted to a limited work range
- it is affected by the load carried by the robot

2.5.3 Repeatability

Repeatability is concerned with the robot's ability to position its wrist or an end effector attached to its wrist at a point in space that had been previously taught to the robot. Repeatability and



accuracy refer to two different aspects of the robot's precision. Accuracy relates to the robot's capacity to be programmed to achieve a given target point. The actual programmed point will probably be different from the target point due to limitations of control resolution. Repeatability refers to the robot's ability to return to the programmed point when commanded to do so. Mechanical inaccuracies are principally responsible for repeatability errors.

Figure 2.16: Illustration of Repeatability and Accuracy

2.5.4 Compliance

The compliance of the robot manipulator refers to the displacement of the wrist end in response to a force or torque exerted against it. Robot manipulator compliance is a directional feature. Compliance is important because it reduces the robot's precision of movement under load.

2.6 End Effectors

The term end effector is used to describe the hand or tool that is attached to the wrist. End effectors can be divided into two categories: grippers and tools. Grippers would be utilized to grasp an object and hold it during the robot work cycle. A tool would be used as an end effector in applications where the robot is required to perform some operation on the workpiece.

2.7 Robotic Sensors

Sensors used in robotics include the following general categories:

- 1) Tactile Sensors - these respond to contact forces with another object and sometimes are capable of measuring the level of force involved.
- 2) Proximity and Range Sensors - Proximity sensors indicate when an object is close to another object but before contact has been made. Range sensors sense the distance between objects.
- 3) Miscellaneous Types - these include sensors for temperature, pressure, and other variables.
- 4) Machine Vision - A machine vision system is capable of viewing the workspace and

interpreting what it sees

Sensors are an important part in work cell control and in safety monitoring systems.

2.8 Robot Programming and Work Cell Control

A robot program can be defined as a path in space through which the manipulator is directed to move. This path also includes other actions such as controlling the end effector and receiving signals from sensors. The purpose of robot programming is to teach these actions to the robot. The two basic methods used for programming robots are leadthrough programming and textual language programming.

Leadthrough programming consists of forcing the robot arm to move through the required motion sequence and recording the motions into the controller memory. This method is used to program playback robots.

Textual programming methods use an English-like language to establish the logic and sequence of the work cycle. A computer terminal is used to input the program instructions into the controller, but a control box (called a teach pendant), used in leadthrough programming, is also used to define the locations of the various points in the workspace. The robot programming language names the points as symbols in the program and these symbols are subsequently defined by showing the robot their locations. Robot languages also permit the use of calculations, more detailed logic flow, and subroutines in the programs, and greater use of sensors and communications.

Work cell control deals with the problem of coordinating the robot to operate with other equipment in the work cell. A robot cell usually consists of not only the robot, but also conveyors, machine tools, inspection devices, and possibly human operators. Some of the activities in the robot work cell occur sequentially, while other activities occur simultaneously. A method of controlling and synchronizing these various activities is required, and that is the purpose of the work cell controller.

2.9 Robot Applications

Industrial applications of robots can be divided into the following 3 categories:

- 1) Material-handling and machine-loading and -unloading applications.
- 2) Processing applications. The function of the robot is to manipulate a tool to accomplish some manufacturing process in the work cell.
- 3) Assembly and Inspection. Inspection robots would make use of sensors to gauge and measure quality characteristics of the manufactured product.

2.10 Jacobians

Suppose we have 6 functions, each of which is a function of 6 independent variables:

$$\begin{aligned}
 y_1 &= f_1(x_1, x_2, x_3, x_4, x_5, x_6) \\
 y_2 &= f_2(x_1, x_2, x_3, x_4, x_5, x_6) \\
 y_3 &= f_3(x_1, x_2, x_3, x_4, x_5, x_6) \\
 y_4 &= f_4(x_1, x_2, x_3, x_4, x_5, x_6) \\
 y_5 &= f_5(x_1, x_2, x_3, x_4, x_5, x_6) \\
 y_6 &= f_6(x_1, x_2, x_3, x_4, x_5, x_6)
 \end{aligned}$$

Vector Notation: $Y = F(x)$

The differential of y_i as a function of differentials of x_j are:

$$\delta y_1 = \frac{\partial f_1}{\partial x_1} \delta x_1 + \frac{\partial f_1}{\partial x_2} \delta x_2 + \dots + \frac{\partial f_1}{\partial x_6} \delta x_6$$

$$\delta y_2 = \frac{\partial f_2}{\partial x_1} \delta x_1 + \frac{\partial f_2}{\partial x_2} \delta x_2 + \dots + \frac{\partial f_2}{\partial x_6} \delta x_6$$

⋮

$$\delta y_6 = \frac{\partial f_6}{\partial x_1} \delta x_1 + \frac{\partial f_6}{\partial x_2} \delta x_2 + \dots + \frac{\partial f_6}{\partial x_6} \delta x_6$$

The above equations are vector notation for:

$$\delta Y = \frac{\partial F}{\partial X} \delta X$$

The Jacobian $J(X)$ is the 6x6 matrix of partial derivatives

$$\begin{bmatrix}
 \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_6} \\
 \dots & \dots & \dots \\
 \frac{\partial f_6}{\partial x_1} & \dots & \frac{\partial f_6}{\partial x_6}
 \end{bmatrix} \equiv J(X)$$

Therefore,

$$\delta Y = J(X) \delta X$$

By dividing both sides by the differential time element, we can think of the Jacobian as mapping velocities in X to those in Y :

$$Y' = J(X) X'$$

At any particular instant, X has a certain value and $J(X)$ is a linear transformation. At each new time instant, X has changed and therefore so has the linear transformation. Jacobians are time varying linear transformations.

In the field of robotics, we generally speak of Jacobians which relate joint velocities to Cartesian velocities of the tip of the arm.

Given that we have a linear transformation relating joint velocity to Cartesian velocity, a reasonable question to ask is: Is this matrix invertible? That is, is it nonsingular? If the matrix is nonsingular then we can invert it to calculate joint rates given Cartesian velocities. This is an important relationship. For example, say we wish the hand of the robot to move with a certain velocity vector in Cartesian space. We could calculate the necessary joint rates at each instant along the path using this relationship. The real question of invertibility is: Is the Jacobian invertible for all values of joint angles? If not, where is it not invertible?

Most manipulators have values of joint angles where the Jacobian becomes singular. Such locations are called singularities of the mechanism. We will class singularities into 2 categories:

1. Workpiece boundary singularities are those which occur when the manipulator is fully stretched out or folded back on itself such that the end effector is near or at the boundary of the workspace.
2. Workpiece interior singularities are those which occur away from the workspace boundary and generally are caused by two or more joint axes lining up.

When a manipulator is in a singular configuration, it has lost one or more degrees of freedom as viewed from Cartesian space. This means that there is some direction (or subspace) in Cartesian space along which it is impossible to move the hand of the robot no matter which joint rates are selected.

APPENDIX A

V+ COMMANDS

PROGRAM INSTRUCTIONS

1.0 Robot Configuration Control

For an anthropomorphic, six-joint robot, most points in its workspace can be reached by assuming one of eight possible spatial configurations. Normally, the robot remains in the configuration in which it starts when the user program begins execution, or continues from a PAUSE instruction or run-time error. The only exceptions are (1) when a READY instruction is executed, (2) when a specific change in configuration is requested by the user program through execution of any of the instructions below, or (3) when the robot is forced to change from FLIP to NOFLIP or vice versa to keep joint 4 or 6 within stop limits.

When the user specifies a change in robot configuration, the requested change is accomplished during execution of the next motion instruction (MOVE, MOVES, APPRO, or DEPART). Note that no configuration change is permitted during a straight-line motion, and that the DELAY instruction has the effect of canceling a pending configuration change.

Note: Since the robot normally does not change configuration when a program begins execution, it is often good practice to begin programs with an initialization sequence that specifies the configuration to be used. (This is especially important when the robot is going to start at the READY location, since the configuration at that location is ambiguous.)

Configuration Commands

RIGHTY or LEFTY

Request a change in the robot configuration so that the first three joints of the robot resemble a human's right or left arm respectively.

ABOVE or BELOW

Requests a change in robot configuration so that the "elbow" of the robot is pointed up (ABOVE) or down (BELOW).

FLIP or NOFLIP

Changes the range of operation of joint 5 to positive (NOFLIP) or negative (FLIP) angles.

2.0 Motion

In the following motion instructions, all distances are to be in millimeters and each “location” argument can be satisfied with a precision point, a transformation, or a compound transformation.

Note that the DELAY instruction (see Section 8. Miscellaneous) has an effect similar to a motion instruction since it is treated as a “move to nowhere” instruction.

Motion Commands

MOVE <location> [!]

Moves the robot to the location and orientation specified by the variable <location>. Intermediate set points between the initial and final robot locations are computed by interpolating between the initial and final joint-variable values, resulting in a joint-interpolated motion. Any changes in configuration requested by the user are executed during the motion.

If the location name is followed by an exclamation point, <location> has its value set, while the instruction is being typed in, to the robot position and orientation at the instant the MOVE instruction is completed by the RETURN key.

Ex: MOVE #PICK!

Moves by joint-interpolated motion to the location described by the precision point PICK, which is defined as the location of the robot when the instruction is completed by a carriage return.

MOVES <location> [!]

Moves the robot to the location and orientation specified by the variable <location>. The tool is moved along a straight-line path and is smoothly rotated to its final orientation. No changes in configuration are permitted during this motion.

Ex: MOVES PLACE

Moves along a straight-line path to the location described by the transformation PLACE.

ALIGN

Causes the tool to be rotated so that its Z axis is aligned parallel to the nearest axis of the world coordinate system. This instruction is primarily useful for lining up the tool before a

series of locations are taught. The simplest way to do this is using the monitor DO command (DO ALIGN).

APPRO <location> [!], <distance>

Moves the tool to the position and orientation defined by the variable <location> and an offset along the tool Z axis of the distance given. A positive distance sets the tool “back” (negative tool-Z) from the specified location. A negative distance offsets the tool “forward” (positive tool-Z).

Ex: APPRO PLACE, 75

Moves the tool by joint-interpolated motion to a location 75 mm from that defined by the transformation PLACE, with the offset along the resultant Z axis of the tool.

APPROS <location> [!], <distance>

Same as APPRO, but the tool is moved along a straight-line path and is smoothly rotated to its final orientation. No changes in configuration are permitted during this motion.

Ex: APPROPS PLACE, -50

Moves the tool along a straight-line to a location 50 mm from that defined by the transformation PLACE, with the offset along the resultant Z axis of the tool and “beyond” the location PLACE.

DEPART <distance>

Moves the tool (by joint-interpolated motion) the distance given along the current Z axis of the tool. A positive distance moves the tool “back”. A negative distance moves the tool “forward”.

Ex: DEPART 80

Moves the tool 80 mm back from its current location.

DEPARTS <distance>

Same as DEPART, but the tool is moved along a straight-line path and is smoothly rotated to its final orientation.

READY

Moves the robot to the READY location above the workspace, which forces the robot into a standard configuration. This instruction always succeeds, regardless of where the robot is originally located.

This instruction is particularly useful with the DO command (DO READY) to check for proper calibration of the robot.

CAUTION: Be careful that the robot does not strike anything while moving to the READY location.

3.0 Variables

Integer and Real

In most cases where an numerical variable is indicated as an argument for a user program instruction, either the symbolic name of the variable can be given or the value can be specified (The only time that a value cannot be used in place of a variable is when the specified operation would result in an attempt to alter the value of the constant; in which case an error message is generated). Variable names conform to the same rules as the location-variable names and program names. Integers must be between $-16,777,216$ and $16,777,215$. Real variables are restricted to the range $\pm 3.4 \times 10^{38}$.

Ex: BOB = 12

Defines (or changes the value) a variable named BOB to have a value of 12

String

A string of alphanumerical characters can be assigned to a variable preceded by a dollar sign. These can be concatenated with other variables, and displayed to the screen using the TYPE command. Each variable name must start with a letter and can contain only letters, numbers, points, and underlined characters. All strings are stored as lower case characters. The content cannot exceed 125 characters, and the variable name must not exceed 15 characters.

Ex: \$reply = "reply by YES or NO"

4.0 Location Assignment and Modification

These instructions define and modify the destinations of robot motion instruction during program execution. It is especially important to understand these very powerful instructions because they provide significant flexibility in V+ programs. For example, by accepting external input signals and then altering the base or tool settings, or defining or modifying the values of location variables, a user program can adapt to the robot working environment.

Location Commands

HERE <location>

Sets the value of a transformation or precision point equal to the current robot location. (This instruction is equivalent to the monitor HERE command.)

Only the right-most transformation of a compound transformation is defined. An error message results if any of the other transformations in a compound transformation are not already defined.

Ex: HERE PART

Sets the transformation PART equal to the current robot location.

Ex: HERE #PART

Assigns the current location of the robot to the precision point PART.

SET <transformation> = <transformation 2>[:<trans 3>] ... [:<trans n>]
or SET <precision point> = <precision point>

Sets the value of the location variable on the left equal to that on the right of the equal sign. If the right-hand side is not defined, an error message is generated.

Ex: SET PICK=START

Sets the value of the transformation PICK equal to that of the transformation START.

Ex: SET #PLACE=#POST

Sets the value of the precision point PLACE equal to that of the precision point POST.

SHIFT (<transformation> BY [<dx>],[<dy>],[<dz>])

Modifies the X, Y, and Z components of the indicated transformation by adding distance changes <dx>, <dy>, and <dz>, respectively. If the value of the transformation has not yet been defined, or it's new X, Y, Z location is too large to represent, an error message is generated.

Ex: SET PICK = SHIFT (PICK BY 100.8,-35.1,0)

Redefines transformation PICK to be shifted 100.8 mm in the X direction and 35.1 mm in the negative Y direction.

TOOL [<transformation>]

Sets the value of the tool transformation equal to the value of <transformation>. Refer to the monitor TOOL command for a description of the effect of this instruction.

Ex: TOOL POINTER

Replaces the current tool transformation assignment with the value of transformation POINTER

INVERSE <transformation> = <transformation 2>[:<trans 3>] ... [:<trans n>]

Similar to the SET instruction. Sets the value of <transformation> equal to the matrix inverse of the transformation on the right. That is, if the transformation on the right represents a location A, relative to another location, B, then the inverse transformation represents the location B relative to A. If the right-hand side is not defined, an error message is generated.

Ex: INVERSE TRANS.INV=TRANS

Defines the transformation TRANS.INV to be equal to the inverse of the transformation TRANS.

FRAME (<trans 1>,<trans 2>,<trans 3>,<trans 4>)

Describes the relationship of a secondary reference frame to that of the robot. The secondary reference frame is assumed to have (1) its X-axis pass through <trans 1> and <trans 2>, (2) its Y axis is a straight line perpendicular to the X axis and its positive direction defined by <trans 3>, and (3) <trans 4> is the origin of the new reference system. This instruction is usually used to define a “base” transformation for relative locations. To improve accuracy, the points should be as far apart as practical.

Ex: SET ref = FRAME (pt1,pt2,pt3,pt4)

Defines the value of the transformation BASE to be a description of the reference frame which has its origin at the point described by transformation A1, with the positive X axis passing from A1 through the point given by A2, and A3 defining another point which lies in the X-Y plane.

5.0 Program Control

The following instructions alter the sequence in which the user program instructions are executed, and interlock the V+ system with other devices.

In this section, “<channel>” designates one of the external signal input or output lines, and can be either an integer number or an integer variable. The absolute value of <channel> specifies a particular signal line. The magnitude of <channel> must be between 1 and n, where n is the number of hardware bi-directional interlocks provided with the system. For certain instructions, the sign of <channel> specifies high or low signal values. A positive value indicates a high signal and a negative value indicates a low signal.

Program Control Commands

GOTO <label>

Performs an unconditional branch to the program step identified by the given label.

CALL <program>

Execution of the current program is temporarily suspended, and execution continues at the first step of the indicated user program, which is then considered a subroutine. Execution automatically returns to the current program when a RETURN instruction is executed in the subroutine. Up to ten programs can be superseded at any given time.

RETURN [<skip count>]

Terminates execution of the current subroutine and resumes execution of the last-suspended program at <skip count> +1 steps (not counting comment instructions) following the instruction which caused the subroutine to be invoked. If the skip count is omitted, or a count less than zero is specified, a value of zero is used. A RETURN in a main program has the same effect as a STOP instruction.

If the subroutine started execution due to triggering of a REACTI instruction (see below), a skip count of one can be used to resume execution of the interrupted routine without executing the interrupted program step again.

Ex: RETURN 2

Returns to the previously suspended program, at the third (2+1) step following the instruction which invoked subroutine execution.

IF <i.var> <relationship> <i.var 2> **THEN** <block1> [**ELSE** <block2>] **END**

Compares the value of <i.var> to the value of <i.var 2>; if the stated relationship is true, program execution branches to the program step identified by the given label. Otherwise, the next step of the program is executed as usual. The possible relationships that can be specified are = (equal), < (less than), > (greater than), >= or => (less than or equal), <= or <= (greater than or equal), and <> (not equal). Note that either variable can be replaced by an integer value. AND, NOT, OR, and XOR (exclusive or) can also be used in

relationships. Else can add further functionality by giving an alternative program flow when the IF relationship is not satisfied.

Ex: IF N > 3 THEN move a

If the current value of the integer variable N is greater than 3, the robot will move to location a, otherwise the program continues with the next step.

IF <var> <relationship> <var 2> **GOTO** <label>

Similar to IF...ELSE...END. Program will jump to <label> if relationship is true.

Ex: IF count >= 7 GOTO 40

FOR <var> = <limit 1> **TO** <limit 2> **STEP** <limit 3> <block1> [**END**]

Loops program through <block1> a specified number of times. <var> will increase (or decrease) from <value 1> to <value 2> in increments of <value 3>. Limits can be constants, variables, or expressions.

Ex: FOR t = 10 TO 1 STEP -2
 MOVE a
 MOVE b
 END

The robot will move to point a and then point b five times in succession.

WHILE (<condition>) **DO** <block1> [**END**]

Executes <block 1> while <condition> is logically true. If the result is false (=0), the program will continue by executing the instructions following the END statement. As the test is performed at the start of the loop, the program step is not obliged to run at least once.

Ex: WHILE (num.part < 12) DO
 MOVE b
 MOVE c
 END

The robot will move between point b and point c until num.parts is greater than or equal to 12.

DO <block 1> **UNTIL** (<condition>)

Executes <block 1> until the result of <condition> is true (=1). This instruction will always execute the block at least once.

Ex: DO
MOVE a
MOVE b
UNTIL (num.parts > 12)

The robot will move between points b and c until num.parts equals 12.

CASE <expression> **OF VALUE** <expression> [, <expression>] : <block 1> **ANY** <block 2>
END

This instruction is used to select a block of instructions to be executed depending on a value contained in the expression. This can be used to make program flow clearer than using a number of IF..THEN..ELSE..END, or IF..GOTO commands.

Ex: CASE t OF
VALUE 1,3,5,7,9:
TYPE "Number is ODD"
VALUE 2,4,6,8:
TYPE "Number is EVEN"
ANY
TYPE "Number is out of range"
END

Console will display whether t is an odd or even number. If t is not an integer from 1-9, the console will indicate that t is out of range.

PAUSE [<string>]

Terminates execution of the user program, and displays the message <string>. Execution can be continued from this point by typing PROCEED.

STOP [<string>]

Terminates execution of the user program unless more program loops (see the EXECUTE command) are to be completed, in which case execution of the program continues at its first step. If provided, the message <string> is displayed on the terminal. The STOP instruction is used to mark the end of a program execution pass. Note that the HALT instruction has a different effect.

HALT [<string>]

Terminates execution of the user program regardless of any program loops (see the EXECUTE command) remaining to be completed, and displays the optional message <string> on the terminal. After termination by a HALT instruction, program execution cannot be resumed with a PROCEED command.

IFSIG <channel> , [<channel>] , [<channel>] , [<channel>] THEN <block>

If the state(s) of the indicated external input signal(s) exactly match the state(s) specified, the program branches to the instruction identified by the given label. That is, if any mismatch is detected, the next program step is executed. Channels are specified as signed values to indicate whether the tests are to be made for high or low signals. Omitted channel numbers or channel numbers with a value of zero always satisfy the matching test.

Ex: IFSIG (2, -3) THEN MOVE a

If the external input signal line #2 is high and line #3 is low, robot moves to point a, otherwise execution continues with the next step.

SIGNAL <channel> , [<channel>, ..., <channel>]

Turns the signal(s) on or off at the specified output channel(s). Positive channel numbers turn the corresponding signals on, negative turn the signals off.

Ex: SIGNAL -1, 4

Turns off the signal at output channel # 1, and turns on the signal at output channel # 4.

REACT <channel>, [<program>] [ALWAYS]

Initiates continuous monitoring of the external signal at the specified input channel. If a high value is detected, the program “reacts” by altering the sequence in which the following program steps are executed. The reaction is equivalent to performing a “CALL <program>” after completion of the program step during which the high signal first occurs. If ALWAYS is specified, the signal monitoring is active until an IGNORE instruction (see below) is executed, or until the reaction is triggered (in which case the equivalent of an “IGNORE <channel> ALWAYS” is automatically performed). If ALWAYS is omitted, the signal is only checked until completion of the next motion instruction. If no program is referenced, the program step following that during which the first high value occurs is skipped and execution continues at the next program step. Only one REACT or REACTI command can be associated with any channel at any given time.

Ex: REACT 3, PICKUP

Monitors external input signal #3; if during the next motion instruction it becomes high, branch to subroutine PICKUP after the motion.

REACTI <channel> , [<program>] [ALWAYS]

Like REACT, this instruction initiates the continuous monitoring of the external signal at the specified input channel. However, in this case when a high value is detected, the current instruction is immediately aborted and the equivalent of a “CALL <program>” is executed. The subroutine call to <program> is performed such that if a “RETURN 0” is encountered, the interrupted program step is executed once again. To return control to the interrupted program and skip further execution of the interrupted step, a “RETURN 1” must be used to exit from <program>. If no program is referenced, the current program step is immediately aborted and the following step is executed when a high signal on the channel is detected. The optional argument ALWAYS and the IGNORE instruction have exactly the same effect on REACTI as they do on the REACT instruction. Only one REACT or REACTI command can be associated with any channel at any given time.

Ex: REACTI 1, ALARM ALWAYS

Monitors external input signal #1; if it ever becomes high, immediately branch to subroutine ALARM.

IGNORE <channel> [ALWAYS]

Disables the REACT or REACTI instruction associated with the specified external-signal input channel. If ALWAYS is specified, the reaction is permanently disabled; otherwise it is disabled only until completion of the next motion instruction. The value <channel> must always be greater than zero.

Ex: IGNORE 5 ALWAYS

Stops monitoring of the external input signal #5.

WAIT <channel>

Puts the program into a “wait loop” until the desired sense of the external signal at the specified input channel is detected. Positive and negative channel numbers indicate waiting should be done until the external signal goes true (high) or false (low), respectively. A signal wait loop can be aborted by using the monitor PROCEED command.

6.0 Trajectory Control

The following instructions are used for the enabling and disabling special features of the hardware position servo and the software trajectory generator. As indicated below, these instructions can include the word ALWAYS if it is desired that the requested option affect all successive motions. Whenever an instruction does not include ALWAYS, it is assumed that it is only to affect the next motion. Note that a DELAY instruction cancels the effects of these instructions when ALWAYS is not specified.

Trajectory Control Commands

SPEED <value> [MMPS] / [IPS] [ALWAYS]

Requests that robot motion be performed at the specified speed, <value>. MMPS denotes that value is in mm/sec. IPS denotes that value is in in/s. If no units are given, then value is considered a percentage of the monitor speed. The robot should not be run above 100% of monitor speed. If ALWAYS is specified, this speed is valid for all movements until a new speed is specified. <value> can be a constant, variable, or mathematic expression, but cannot be less than 0.000001

Ex: SPEED 60 ALWAYS

Sets the program motion speed to 60% of monitor speed until changed by another SPEED command.

7.0 Miscellaneous

The following instructions make a variety of capabilities available to the programmer.

BASE [<dX>],[<dY>],[<dZ>],[<Z rotation>]

Redefines the reference frame of the robot in the same way as the monitor BASE command.

Ex: BASE 100 , , -50

Redefines the world reference frame to effectively shift all locations 100 mm in the negative X direction and 50 mm in the positive Z direction from their current location. Note that the arguments for this instruction describe the movement of the robot reference frame relative to its present location, and thus have an opposite effect on locations relative to the robot.

DELAY <time>

Puts the program into an idle loop for the specified period of time. The duration is interpreted in seconds and must be greater than 16 milliseconds.

It should be noted that DELAY is interpreted as a “move-to-nowhere” motion instruction. In particular, (1) if there is a pending hand operation, the hand motion takes place during execution of the DELAY instruction; (2) if any temporary trajectory switches have been specified, they are cleared after the conclusion of the DELAY; and (3) if there is a pending configuration change, it is canceled.

Ex: DELAY 2.5

Causes all robot activity to stop for 2.5 seconds, any pending hand operation occurs, clears any temporary trajectory switches that may be set, and cancels any pending configuration requests.

ENABLE <switch> or DISABLE <switch>

These instructions have the same effects on the system as the corresponding monitor commands.

Ex: ENABLE MESSAGES

Turns on the MESSAGES switch.

; [<string>] (*semicolon)

Provides a “comment” line in a program. That is, any line beginning with a semicolon is used only for the programmer’s benefit and is ignored when a program is executed.

TYPE [<string>]

Displays the message <string> on the terminal; a blank line is output if no string is provided. If the MESSAGES switch is DISABLE’d, no output is done.

PROMPT [<string>], <var> [, <var>, <var>,...]

Used to input information from the user. The program will be suspended until the operator answers. The reply can be a real variable or variables and/or a string of characters. If the user replies to several parameters, they must be separated by commas.

Ex: PROMPT “Give me the programmer’s name” , \$name

Program will ask for the user’s name, wait until something is entered at the terminal, and store the input in the variable name.

SAMPLE PROGRAMS

The following examples are intended to demonstrate the use of various V+ instructions. The tasks used in the examples have been chosen because they are representative of common types encountered.

1. Program Initialization

It is often useful to begin a program with an initialization sequence to make sure certain conditions are as required for proper execution of the program. The following example establishes the configuration of the robot and outputs, and makes sure the hand is open.

Since one never knows when it might become desirable to have a program invoked as a subroutine by another program, it is good practice to terminate a program with a RETURN instruction unless a STOP or HALT is required for some reason.

```

;           START OF PROGRAM TO ...
;
;           ASSURE THE PROPER ROBOT CONFIGURATION
RIGHTY
ABOVE
NOFLIP
;
;           ASSURE CORRECT INITIAL HAND OPENING
OPENI 100.0
SPEED 50 ALWAYS
SIG 57, -32
;
;           START OF TASK INSTRUCTIONS
.
.
;
;           END OF TASK INSTRUCTIONS
;
;           END WITH "RETURN" SO THIS ROUTINE CAN BE USED AS A
;           SUBROUTINE LATER IF DESIRED
RETURN          0
;
;           -- END OF PROGRAM --

```

2. Palletizing

This example demonstrates the use of the SETI, IF, SET, and SHIFT instructions. The task to be accomplished is to pick objects from a pallet with six rows (50 mm apart) and 12 columns (30 mm apart) of bins. The instructions shown here only define the locations of successive objects. Other instructions would be included in the indicated space to perform the desired operations on the objects.

The plane of the pallet is assumed to be parallel to the X, Y, plane of the robot world coordinate system. The rows are assumed to have a direction 30 degrees from the normal X axis, as indicated by the BASE instruction below. If these assumptions are not correct, the arguments in

the SHIFT instructions would have to be changed to represent the correct X, Y, Z displacements between pallet positions.

A “main” program that will administer this task follows:

```

;          PROGRAM TO PICK OBJECTS FROM A PALLET
;
;          INITIALIZE VARIABLES FOR THE PALLET SUBROUTINE
;
;          SET THE REFERENCE FRAME AXES PARALLEL TO PALLET
BASE 0.00, 0.00, 0.00, 30.00
;          SET VALUES OF MAXIMUM ROW AND COLUMN COUNTS
MAX.ROW = 6
MAX.COL = 12
10 ;          SET ROW COUNTER TO FIRST ROW
    ROW = 1
;          SET COLUMN COUNTER TO FIRST COLUMN
    COLUMN = 1
;          SET OBJECT LOCATION TO TAUGHT LOCATION IN THE PALLET
    PICK = CORNER
20 ;          INSTRUCTIONS DESCRIBING MOTIONS TO AND FROM LOCATION
;          “PICK” AND ACTIONS TO BE PERFORMED TO THE OBJECT
;
;          .
;          .
;          .
;          REDEFINE “PICK” TO BE NEXT PALLET LOCATION
CALL PALLET()
;          GO BACK AND PROCESS NEW PALLET LOCATION
GOTO 20
;          GET HERE WHEN WHOLE PALLET HAS BEEN PROCESSED
;          BECAUSE OF “RETURN 1” IN SUBROUTINE
;          RETURN TO CALLING PROGRAM OR MONITOR
RETURN 0
;          -- END OF PROGRAM --

```

The subroutine PALLET will define the successive values for location variable PICK

```

;          SUBROUTINE PALLET
;
;          PURPOSE:  CALCULATE PALLET LOCATIONS
;
;          INCREMENT COLUMN COUNTER
COLUMN = COLUMN + 1
;          TEST FOR END OF ROW
IF COLUMN > MAX.COL GOTO 10
;          SHIFT ALONG ROW
SET PICK = SHIFT (PICK BY 30.00, 0.00, 0.00)
;          RETURN TO MAIN PROGRAM AND PROCESS NEXT OBJECT
RETURN 0
;
10 ;          ROW COMPLETED, INCREMENT ROW COUNTER
    ROW = ROW + 1

```

```

;           TEST FOR ALL ROWS DONE
IF ROW > MAX GOTO 20
;           SHIFT BACK TO START OF ROW, AND DOWN TO NEXT ROW
SET PICK = SHIFT (PICK BY -330.00, 50.00, 0.00)
;           RESET COLUMN COUNTER
COLUMN = 1
;           RETURN TO MAIN PROGRAM AND PROCESS NEXT OBJECT
RETURN 0
;
20 ;           ALL PALLET POSITIONS HAVE BEEN USED, RETURN TO MAIN
;           PROGRAM AND SKIP THE "GOTO" INSTRUCTION
RETURN 1
;           -- END OF SUBROUTINE --

```

Before these routines can be used, location CORNER must be defined, as well as any locations referenced by the omitted instructions. Location PICK is redefined to correspond to each pallet location during execution. If the pallet is later moved, only location CORNER need be redefined -- all the other pallet locations are calculated by PALLET.

3. Communicating with External Signal Lines

This sample program performs the following task: (1) the robot waits for a part to be in place in a feeder; (2) after picking up the part, the robot carries it to an inspection station, and signals the station that a part is in place; (3) the station determines whether the part is type "A" or "B" and sets the states of signal lines accordingly; (4) based on the output of the inspection station, the robot is directed by one of three subroutines to process the part; (5) the cycle repeats indefinitely.

The program will immediately branch to subroutine EMERGENCY if an emergency condition is indicated by input signal line #7 at any time from the start of the program until the IGNORE instruction.

```

;           START OF PROGRAM
;
;           INITIALIZE SIGNAL LINE
SIGNAL -2
;           MAKE SURE HAND IS INITIALLY OPEN
OPENI 100.00
;
10 ;           START OF LOOP TO PROCESS PARTS
;
;           START LOOKING AT "EMERGENCY" SIGNAL ON INPUT
;           CHANNEL 7
REACTI 7, EMERGENCY ALWAYS
;           WAIT FOR "PART IN PLACE" SIGNAL ON INPUT CHANNEL 1
WAIT 1
;           PICK UP PART FROM FEEDER
SPEED 200.00
APPRO PART, 50.00
MOVES PART

```

```
CLOSEI 0.00
DEPARTS 50.00
;
APPRO TEST, 75.00
MOVES TEST
;
IGNORE 7 ALWAYS
;
SIGNAL 2
;
WAIT 6
;
DEPART 100.00
;
SIGNAL -2
;
IFSIG (-3, 4, -5) GOTO 20
;
IFSIG (3, -4, -5) GOTO 30
;
CALL REJECT
GOTO 40
20 ;
CALL PART.A
GOTO 40
30 ;
CALL PART.B
40 ;
GOTO 10
;
-- END OF PROGRAM --
```

The inspection station is assumed to perform tests that identify the part. The results are reported to the robot by the status of input signal lines 3, 4, and 5.

4. Use of Tool Transformations

Suppose that in the last example the parts are to have a hole drilled in them, and it is to be in different places on the two parts. Thus, the part processing involves holding the parts under a drill in the proper position and orientation.

One way to program this task would be to have both subroutines PART.A and PART.B contain identical motion sequences, with the only difference being the locations involved. This approach will work, but it may be undesirable because of the duplication of instructions required. Also, this approach will get very complicated if several other work stations are involved.

Another way to program the task is to use the same instruction sequence for both parts, and use tool transformations to compensate for the differences between the parts. The instructions to do this might be as follows (omitted instructions are the same as in the preceding section).

```

      .
      .
      .
;          CALCULATE THE TOOL TRANSFORMATIONS
INVERSE   PART.A = INV.A
INVERSE   PART.B = INV.B
10 ;      START OF LOOP TO PROCESS PARTS
;
;          SET THE TOOL TRANSFORMATION FOR THE NULL TOOL
;
TOOL
      .
      .
      .
20 ;      PROCESS PART "A"
TOOL PART.A
GOTO 40
;
30 ;      PROCESS PART "B"
TOOL PART.B
;
40 ;      PROCESS THE PARTS
APPRO DRILL, 100.00
MOVES DRILL
;          SIGNAL THAT PART IS UNDER DRILL
SIGNAL 3
;          WAIT FOR "DRILL DONE" SIGNAL
WAIT 6
;          WITHDRAW FROM THE DRILL AND DROP OFF PART
DEPART 100.00
MOVE RELEASE
OPEN 100.00
DEPART 80.00
;          GET ANOTHER PART
GOTO 10
;          -- END OF PROGRAM --

```

Note that both types of part will be moved through the same sequence. The following sequence should be used to define the necessary transformations:

1. Move the empty hand to a location that is fixed relative to the drill. (This location should be well defined so the hand can be returned to it exactly if the drill should later be moved.)
2. Type "HERE DRILL" to define location DRILL.
3. Put a type-A part in the hand and position it under the drill.
4. Type "HERE DRILL: INV.A" to define the transformation INV.A.
5. Put a type-B part in the hand and position it under the drill.
6. Type "HERE DRILL:INV.B" to define the transformation INV.B.

7. Move to the drop-off location and type “HERE RELEASE”.

Note that the transformations INV.A and INV.B are the inverses of the tool transformations used in the program. The INVERSE instructions in the program calculate the tool transformations from these transformations.

Once the transformations PART.A and PART.B are defined, the relationships between the robot and the two hole locations are established. This could be especially useful if the parts are to be processed at several machining stations. To teach the locations corresponding to other stations, the following sequence could be used:

1. Type “TOOL PART.A” to set the tool transformation for part A.
2. Put a type-A part in the hand.
3. Position the part at the desired work station.
4. Type “HERE <station name>” to record the location.
5. Repeat steps 3 and 4 for the other stations.

This not only teaches the locations for processing type-A parts, but also automatically includes type-B parts. This is because when type-B parts are processed, the tool transformation PART.B correctly represents the relationship between the hand and the hole in those parts.