01 Aug 1990

Robot Pedagogics: The Adaptation, Analysis, and Computer Control of a Model Manipulator

Edward T. Hammerand

Chung You Ho
Missouri University of Science and Technology

ROBOT PEDAGOGICS: THE ADAPTATION, ANALYSIS, AND COMPUTER CONTROL OF A MODEL MANIPULATOR

E. T. Hammerand* and C. Y. Ho

CSc-90-5

Department of Computer Science

University of Missouri-Rolla

Rolla, Missouri 65401 (314)341-4491

*This report is substantially the Ph.D. dissertation of the first author, completed August 1990.

PUBLICATION THESIS OPTION

This dissertation has been prepared in the style utilized by the *Communications of the ACM*. It has been accepted for publication as a book in the Computer Science and Computer Engineering series published by Ablex Publishing and edited by Dr. George W. Zobrist.

ABSTRACT

The subject of robotics is addressed by many different fields, among them computer science, electrical engineering, and mechanical engineering. This work is an attempt to bring together all of these aspects from the perspective of a computer science background. Different techniques are considered and reconciled with one another in the analytical area, while detail and explanation are added in all areas that were not previously available. In addition, geometrical interpretations are presented for concepts that have heretofore been presented only in the form of equations.

TABLE OF CONTENTS

Page

Page

Page

LIST OF ILLUSTRATIONS

Figure Page

Figure Page

LIST OF TABLES

I. INTRODUCTION

The subject of robotics is addressed by many different fields, among them computer science, electrical engineering, and mechanical engineering. In particular, the construction and implementation of a robotic manipulator are dealt with in mechanical and electrical engineering, while the control of the manipulator falls more within the realm of computer science. This work examines the different aspects of robotics using a simple model manipulator. The emphasis here shall be from the mathematical and computer control perspectives.

A. OVERVIEW OF TOPICS

There are many types of robots in use today, as evidenced in the surveys of [Zeld84] and [Card85]. Examples of these manipulator types include the Armstar, ASEA, KUKA, Mobot, Puma, Cincinnati Milacron, Prab, Devilbliss, IBM RS 1, and Seiko. These robots are designed for use in production environments and range in cost from a few thousand dollars to in excess of one hundred thousand. There are also smaller, less expensive arms manufactured for private use, ranging from several thousand dollars to just a few hundred. The robot arm utilized for this work is less expensive still; it is the Armatron manipulator, manufactured for Radio Shack. While the Armatron is not suitable for practical applications, it serves well to illustrate the concepts, techniques, and limitations involved in robotics.

As manufactured, the Armatron manipulator is intended for direct manual control only. Chapter 2 of this work details the mechanical and electronic adaptation of the manipulator for computer control. Sufficient work is performed to provide for the computer direction of individual joints one at a time. Several limitations will be immediately apparent with this design. For example, as no feedback is implemented on the arm, positioning will not be precise. Works such as [Saff79] and [Robi84] consider topics such as feedback and velocity control. The implementation of aspects such as these is neglected in favor of more detail in the areas of mathematical analysis and computer control.

Manual control of the robot arm via the computer keyboard is achieved in Chapter 3. With the manipulator adapted for computer control, this step comes rather easily. The computer keyboard is used in a fashion analagous to that of the teach pendant in a production robot environment; various keys are used to directly control the movement of the robot joints. Were this a production environment, safety aspects such as those discussed in [Nico85] would have to be considered. The program control developed implements a dead man switch feature; this is the most basic of precautionary measures that need to be taken when the strength of the robot arm is sufficient to cause injury or death.

Computer control is developed for the manipulator by first establishing the position and orientation of the end of the manipulator with respect to some base coordinate frame. Denavit and Hartenberg put forth a representation for the solution of this problem in 1955 that became the standard for robotics [Dena55]. Chapter 4 develops a control structure based on the variables associated with the manipulator joints, or robot articulation variables as they are referred to in [Coif83b]. [Paul81c] and [Paul87], among others, carry out the mathematical process in lesser detail for a number of different manipulators. A new interpretation of the position and orientation provided under the Denavit and Hartenberg representation is presented in the course of the development here.

Once the relationship of position and orientation from joint variables is established, the inverse relationship may be pursued: the determination of joint variable values required to achieve a specific position and orientation of the gripper. This is the topic of Chapter 5; trigonometric equations resulting from the Denavit and Hartenberg representation must be solved for the angles involved. The first general approach to this problem was put forth by Paul [Paul81a]; it was then refined by Ho and Copeland [Ho82] and later Ranky and Ho [Rank85] to utilize a decomposition of the entire manipulator into an arm proper and a wrist to bring about simplification of the trigonometric equations involved. Others, including [Lee82], [Feat83], and [Gold85] have addressed this problem as well, but none puts forth a more workable solution. An examination of a simplified manipulator is presented in [Brad83], which demonstrates the basic principles involved. The problems associated with manipulator wrists are presented by [Holz86]. The wrist of the Armatron consists of only two degrees of freedom, where three degrees are required in order to attain any orientation. A new geometric interpretation of the inadequacies present in such a wrist is presented here.

Velocity control for the Armatron manipulator is considered in Chapter 6. Velocity may be discussed in terms of the rates at which the gripper moves and turns or in terms of the rates at which the robot joints turn. The velocity control problem is concerned with defining one type of rates in terms of the other; this is done in both directions. Paul presents an approach to the problem which results in gripper, or end effector, rates with respect to a coordinate frame fixed with respect to the moving end effector in [Paul81b] and [Paul81c]. Ho and Sriwattanathamma present a different technique which results in end effector rates with respect to a stationary base frame in [Ho86] and [Ho90]. The result of the latter approach is more naturally interpreted. Some effort is devoted to the reconciliation of the approaches here; the results are shown to be equivalent. In the determination of joint rates, Paul uses differentiation of trigonometric equations while Ho and Sriwattanathamma utilize a matrix algebra method. The determination of joint rates required to achieve some desired set of translational and rotational rates is the more useful of the two directions, and more emphasis is placed on it here. The two approaches are shown to produce equivalent results; the advantages of each are also pointed out. Singular conditions are identified for the techniques as well; Spong also addresses this problem [Spon89].

The final area to be considered in this work is trajectory control, that of guiding the manipulator along a desired path. Paul provided one of the first comprehensive solutions to this problem as well [Paul79]. Grossman had earlier pointed out that most of the development to date had utilized teaching methods to specify what positions the manipulator gripper should pass through and noted the benefits that awaited from a more general software control [Gros78]. Takase and Paul also developed a teaching approach for trajectory control [Taka81]. Others, such as [Coif83a], [Khal84], [Kirc85], and [Pare85], discuss trajectory control from a solely mathematical perspective; desired points or joint settings are specified as part of the problem definition and solutions are mathematically developed to satisfy the requirements. Cook and Ho [Cook82] present an algorithmic approach to the problem, which was later expanded on by Ranky and Ho [Rank85]. This same technique is developed here and further expanded upon to include critical position testing prior to the trajectory traversal; if the trajectory derived from the specified path nodes will lead the manipulator toward an unattainable position, it should not be attempted.

B. THE COMPUTER PROGRAM

For each of the topics to be covered, analytical derivations and numerical examples will be provided; in addition, a computer program is developed in parallel with the analysis and example usage. The programming language chosen for this task is Borland's Turbo C, the selection of which was motivated in part by the nature of the control problem at hand and C's relationship with assembler language. Advantage is taken of C's built-in procedures that fall under the scope of control usage. On the other hand, little use is made of some of C's more exotic mathematical features, as the desire is to keep the program logic and structure as clear and easy to follow as possible.

The main procedure of the program begins with some initialization tasks. First, the control lines leading to the robot are cleared. Next, a query is issued as to whether the displays generated during the course of program execution should be saved to disk. The query is part of the introductory screen generated under direction of the main procedure and depicted in Figure 1.1. The arrays containing the joint variable values and orientation and postion matrix are also initialized here; these will be fully explained in Chapter 4. The body of the main procedure follows:

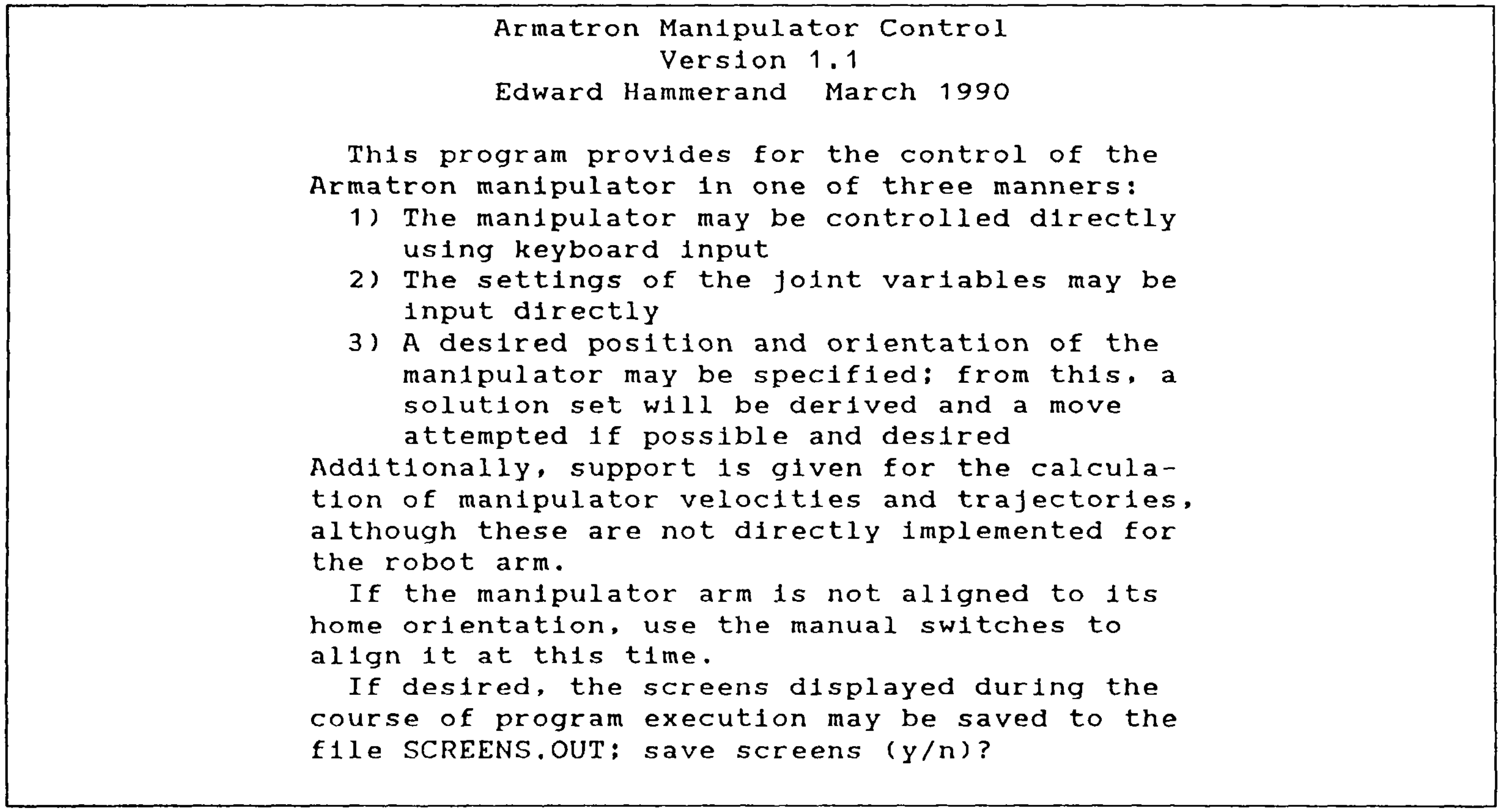
```
outportb (888, 0);
fptr = fopen ("SCREENS.OUT", "w+t");
dsply_main_introduction ( );
locate (23, 55);
qsave_screen = toupper (getch ( ));
lputch (23, 55, qsave_screen);

wait_then_erase (1);
dsply_thetas_noap (&row, cols);
for (i = 1; i <= 5; i++)
   theta[i] = 0;
noap_matrix (theta, noap, row, cols);
dsply_main_selection ( );
while ( (opt = get_option(5)) != 0 )
   {
   wait_then_erase (8);
   switch (opt)
      {
      case 1 : manual_control (theta, noap, row, cols);
               break;
      case 2 : joint_variable_control (theta, noap, row, cols);
               break;
      case 3 : position_orientation_control (theta, noap, row, cols);
               break;
      case 4 : velocity_control (theta, noap, row, cols);
               break;
      case 5 : trajectory_control ( );
               break;
      }
   dsply_main_selection ( );
   }
fputc (eof, fptr);
fclose (fptr);
```

The available control methods are then presented in the display of Figure 1.2. The procedure subsequently iterates, passing control to the selected control method; Chapters 3, 4, 5, 6, and 7 will fully explain the procedures associated with the manual, joint variable, position and orientation, velocity, and trajectory control methods, respectively. The procedure concludes by closing the disk file to which the screens may have been saved.

A final word needs to be said here about the C programming style employed. Full function prototypes and parameter lists are used to provide the compiler with as much information as possible. The complete set of function prototypes for all of the procedures to be used in this work may be found in Appendix A, along with all defined types, declared constants, and include files for the referenced built-in C functions. Due to the amount of program source code involved, a single source file for the entire program would be of excessive size. The program was thus assembled as a Turbo C project, wherein the source code for each of several separate compilations resides in a separate file. The contents of the project file ROBOT.PRJ are as follows:

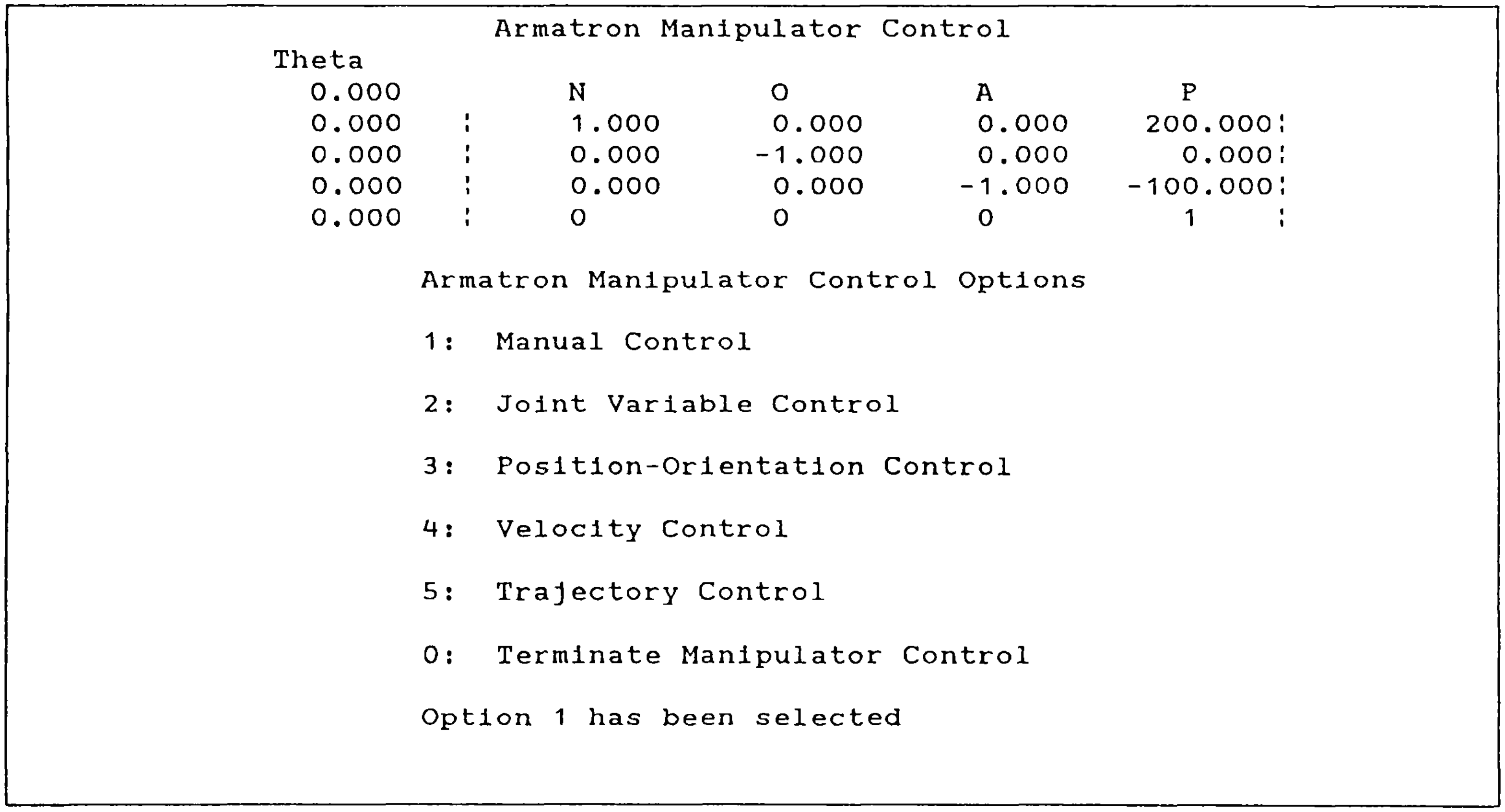
```
main.c
manual.c
jointvar.c
posorien.c
velocity.c
traject.c
```

Each of the control methods, as well as the main procedure and its subordinate procedures, was maintained in a separate source code file as part of the project. Each of these files begins with the #include directive to include the file HEADER.C as part of each separate compilation. In this way function prototyping was supported while simultaneously allowing procedures of one source file and compilation to reference those of another. The documented listing for the procedures associated with the main procedure may be found in Appendix B; subsequent chapters will reference the appendices containing the relevant source code.



```
                 Armatron Manipulator Control
                          Version 1.1
                 Edward Hammerand  March 1990

  This program provides for the control of the
Armatron manipulator in one of three manners:
  1) The manipulator may be controlled directly
     using keyboard input
  2) The settings of the joint variables may be
     input directly
  3) A desired position and orientation of the
     manipulator may be specified; from this, a
     solution set will be derived and a move
     attempted if possible and desired
Additionally, support is given for the calcula-
tion of manipulator velocities and trajectories,
although these are not directly implemented for
the robot arm.
  If the manipulator arm is not aligned to its
home orientation, use the manual switches to
align it at this time.
  If desired, the screens displayed during the
course of program execution may be saved to the
file SCREENS.OUT; save screens (y/n)?
```

Figure 1.1. Main Procedure Introduction



```
                         Armatron Manipulator Control
              Theta
                0.000            N           O           A           P
                0.000   :      1.000       0.000       0.000     200.000:
                0.000   :      0.000      -1.000       0.000       0.000:
                0.000   :      0.000       0.000      -1.000    -100.000:
                0.000   :      0           0           0           1    :

                     Armatron Manipulator Control Options

                     1:   Manual Control

                     2:   Joint Variable Control

                     3:   Position-Orientation Control

                     4:   Velocity Control

                     5:   Trajectory Control

                     0:   Terminate Manipulator Control

                     Option 1 has been selected
```

Figure 1.2. Main Procedure Selection Display

II. CONSTRUCTION OF THE ROBOT

As mentioned previously, the goal of this work is a tutorial examination of robotics by example, both in hardware and software. To this end, this chapter traces the hardware steps and development process in the creation of a scale-size robot arm. This will be done in two phases, mechanical and electronic. The electronic portion of the chapter goes into some detail in an attempt to explain precisely the hows and whys of each step taken. Some insight is provided along the way into some fundamental aspects of computer electronics, such as transistors and logic gates. This chapter is based on the work of Banas [Bana85]. Several problems were determined and corrected in the original work, some of which called for minor modifications and others re-design.

A. MECHANICAL CONSIDERATIONS

To obtain a robot arm for this project, two choices presented themselves: either construct a suitable arm from scratch or adapt an already existing arm to meet the requirements of the project. The complete construction of a robot arm from the ground up is a major endeavor, with many formidable problems to be overcome. The second alternative was realized in the form of an Armatron robot arm, distributed by Radio Shack.

1. The Robot Arm Proper. Although not marketed as such, the Armatron is a five-degrees-of-freedom robot arm, with additional control over its gripper. The Armatron provides a realistic scale-model version of industrial robots of similar design. At the time this project began, the Armatron arm had the additional advantages of being relatively inexpensive and readily available. Unfortunately, Radio Shack has discontinued this version of the arm, replacing it with a newer model containing fewer

degrees of freedom. The robot arm, along with the completed electronic interface, is depicted in Figure 2.1.

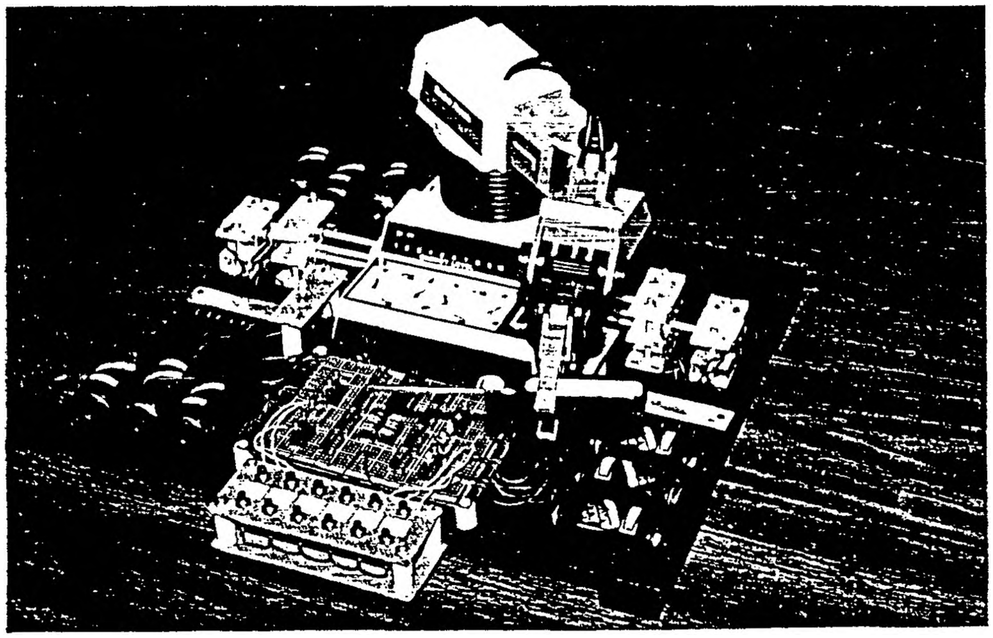


Figure 2.1. The Completed Robot Arm

One of the unusual features of the Armatron is the manner in which it drives its joints. As manufactured, a single 6-volt DC motor turns a shaft containing six gears. The Armatron has a pair of joystick-like levers which may each be pushed forward and back, moved left and right, and twisted clockwise or counterclockwise. Each of these motions engages a different gear arrangement connecting a specific joint with the rotating set of gears. The re-design of the Armatron arm for computer control calls for its lever-operated gear system to be removed and replaced. In order to obtain computer control over the individual joints, one motor will be used for each joint. An electronic interface will be constructed in the next section to direct the motors by computer. Each motor will deliver power to its respective gear by means of a shaft extending from the gear to the motor outside the Armatron housing. Finally, the gears and shafts will be supported at the proper mesh positions by bearing blocks, while the motors are held at respective positions by support blocks.

2. Motors. Due to the relatively close spacing of the required gear positions inside the Armatron, the motor spindles themselves have to be correspondingly close; this in turn calls for small motors. After checking with many sources, Radio Shack was found to have motors with two flattened sides, thus providing a sufficiently short height. The flattened sides also served to make the motors easy to block. The only drawback to the motors is that they were rated for only 1.5 to 3 volts, but the motors and voltage limitation were determined adequate. The shaft of a motor extends approximately one-fourth of an inch and is about one-sixteenth inch in diameter.

It is worth noting here that while the original design of the Armatron required that the new motors be placed outside the housing of the arm as no room was provided on or in the arm itself, this can actually work to an advantage for robot control. Any amount of weight, or more properly mass, that the arm must support increases the required strength and rigidity of the arm. If this model were to be scaled up, the placement of the motors off of the arm itself would simplify the engineering task with regard to these criteria. Further, the complicated problem of dynamic control is lessened somewhat as the inertia of the arm becomes less of a concern due to the decrease in mass.

3. Shafts. To deliver power from the motor spindle to the gear requires a gear shaft. The spindles of the motors used have the one-sixteenth inch diameter dimension to contend with as well as being grooved. (As purchased the motors had small gears, also grooved, press-fit onto their spindles.) Various hobby and model railroading shops were found to carry a type of plastic tubing for use in model construction. The tubing is manufactured in various diameters and has a hollow center. While not being completely rigid, the tubing does not give very much and was judged to be at least as strong as required. The one-eighth inch diameter tubing's center was just large enough

to force over the motor spindles; in fact, the grooves on the spindle notched the inside of the tubing as it was forced on. This created a tight fit between the spindle and the tubing; the motors were found to stall before they would slip inside the tubing.

4. Gears. With the motors and shafts ready, the gears were needed next. Six gears were removed from the original lever-operated system. Each had to have a hole drilled in it so that the shaft could be force fit inside it. Three drill bits of increasing size up to one-eighth inch were used, taking care to keep the center of the gear at the center of the widening hole. Test fits of the shafts to gears found some pieces of tubing to be slightly narrower than others; some fits allowed the gear to spin without much force while others fit so tightly once on that they could not be moved on the shaft without great difficulty. Test fits were made until six sufficiently tight grips were obtained. Simple friction was all that appeared necessary for the fit to hold, so no adhesive was used. Had it been needed, adhesive would have been difficult to apply inside the bearing blocks that will be used to support the gear-shaft combinations. An assembled combination of motor, shaft and gear may be seen in Figure 2.2.

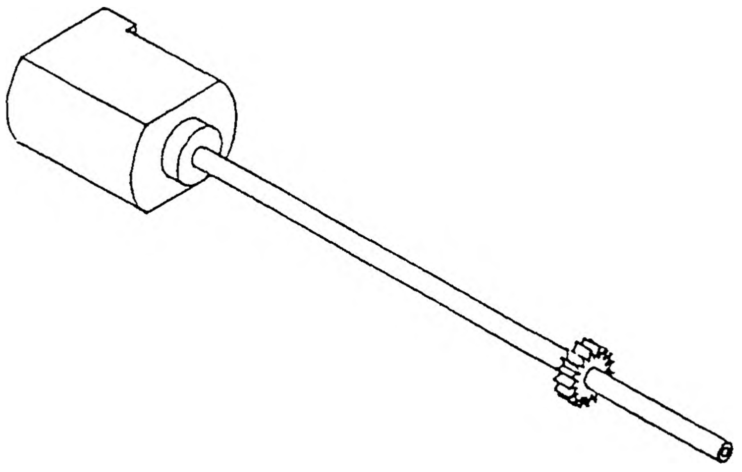


Figure 2.2. Motor, Shaft, and Gear Assembly

5. Bearing Blocks. The most difficult step in the physical alterations made to the Armatron arm was the manufacture of the bearing blocks for the gears and shafts. With the original gearing system removed, there was room in the housing for two approximately 1" x 1" x 2" blocks, each of which would support three gear-shaft combinations. The difficulty lies in the fact that the three points of gear-to-gear meshing required of a block have to all be met within a narrow tolerance. The blocks obtained were manufactured from transparent plastic in a machine shop. Numerous test fits and adjustments were required before all six gears meshed properly. The blocks as cut fit so tightly in the housing that no restraints were deemed necessary.

6. Motor Support Blocks. With the bearing blocks holding each gear and shaft pair in its proper position, the motors could be located outside the housing. First, the gear shafts were cut off long enough to allow some slight deviations in motor alignment. As the shafts were not completely rigid, a slight rise or drop from the bearing block across the shaft to the motor would not cause it to bind and prevent it from operating properly. Support blocks were cut to hold the motors as close as was possible to determine, and then paper shims were used to raise and lower the motors until a smoothly rotating shaft could be observed when power was supplied. One bearing block, the three corresponding motors, shafts, gears, and motor support blocks may be seen in Figure 2.3. At this point, the mechanical alterations were complete and the arm was ready to have the electronic interface set up.

B. THE ELECTRONIC INTERFACE

As described in the mechanical phase of the construction, six motors are required for the five joints and the gripper of the robot arm. An electronic interface between

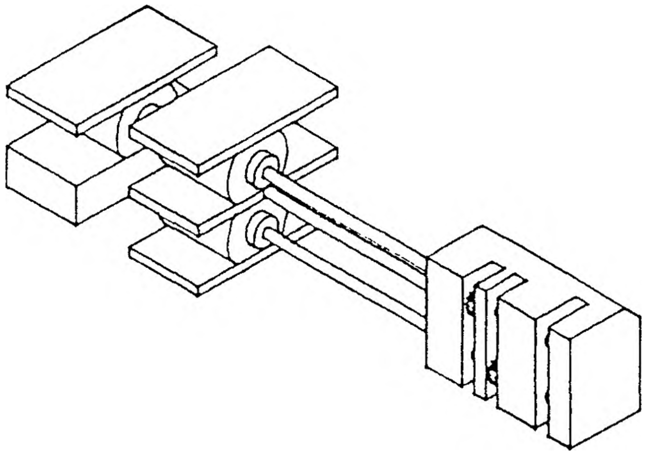


Figure 2.3. Bearing Block to Motor Assembly

the motors and a set of computer output lines will provide the desired computer control.

1. Power Supplies. First consider the motor side of the interface. Each motor is connected by a shaft to one gear arrangement which in turn drives a single joint. The motors are, as mentioned previously, reversible so that the joint can be driven in either direction. For one direction of turn, a motor must have a voltage on one of its lines approximately 3 volts higher than that of the other line. In order to utilize the reversibility of the motor, the line that was 3 volts higher than the other line before must now be approximately 3 volts lower. An arrangement allowing this selection of voltages requires first two independent 4.5 volt power supplies. (The discrepancy in voltage will be explained later.) The power supplies are linked serially, providing a relative ground, a point at 4.5 volts, and a third point at 9 volts. One line from the motor is connected directly to the 4.5 volt location. When the other line is connected to the ground, the motor turns in one direction; when the line is connected to the 9 volt point, the 4.5 volt difference is again in effect but now reversed, causing the motor to turn in the opposite direction. The actual circuitry for the power supplies will be given

here with only an introductory explanation so that the fundamentals behind the circuit elements may be explained in the context of the computer control circuit.

Figure 2.4 shows the circuit which will provide power for the motors actuating the joints. Note that switch S1 is a double-pole, single-throw (DPST) switch, meaning that the circuit is opened and closed at the two places indicated simultaneously by the switch. When the switch is closed, the point labeled as +9 volts is seen to be separated from the point labeled as +4.5 volts by the three 1.5 volt batteries B1, B2, and B3. Similarly, the +4.5 volt point is separated from the point labeled ground by another set of three 1.5 volt batteries, B4, B5, and B6. The light-emitting diode LED1 and resistor R1 complete a circuit with B1, B2, and B3, which is useful in that LED1 glows indicating that switch S1a is closed and B1, B2, and B3 are not dead. LED2 and R2 provide a similar function for B4, B5, and B6. The action of the LED and the resistor will be explained in more detail later in the chapter.

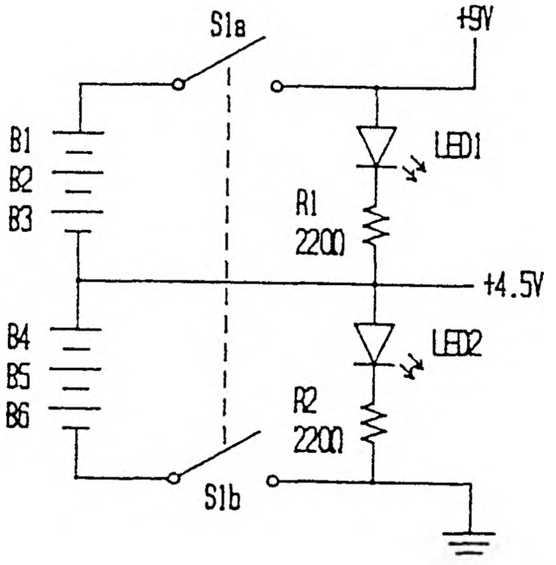


Figure 2.4. Circuit for Motor Power Supplies

One other power supply is needed to power the integrated circuits which will be used in the electronic interface. As will be seen, all of the circuits to be used will require a +5 volt supply. As it turns out, +4.5 volts is sufficient, so the circuit to be built is essentially one-half of the motor power supply; see Figure 2.5. The switch used is of the single-pole, single-throw (SPST) variety as only one point is to be closed. The batteries, LED, and resistor are identical to those used in the previous circuit.

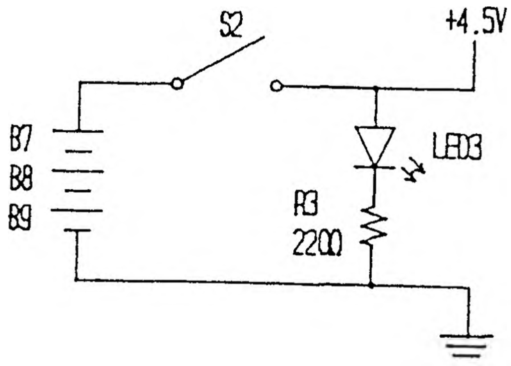


Figure 2.5. Circuit for Integrated Circuit Power Supplies

2. Transistors. In order to allow current to flow through the motor to ground or from the 9 volt source, a switching mechanism is required. The transistor is made for this purpose. To understand how the transistor functions, one must begin with some basic concepts in the chemistry and physics of the atom.

a. Intrinsic Semiconductors. Recall that the elements may be classified as one of three general types according to their electrical conductivity. First are the conductors, such as iron, nickel, and copper; electric current flows easily through these elements. At the other end of the spectrum are the insulators, including carbon, nitrogen, and phosphorous; very little electric current can be made to flow through these elements. Between these two extremes is the third group, semiconductors, encompassing such elements as boron, silicon, and germanium; semiconductors will conduct electric current, but they do so only poorly. In order for electric current to be able to flow in any material, there must be electrons present which are capable of moving through the material. For any given element, electrons may be found orbiting each atom present; further, the number of electrons per atom is equal to the atom's number of protons, which are oppositely (positively) charged and found in the atom's nucleus. The remaining atomic particle is the neutron, which bears no charge and consequently has no direct influence over the flow of electric current through the element.

For an electron to be capable of leaving the atom with which it is associated and moving away the forces which hold it in place must be surmountable. The only electrons that are generally capable of leaving at all are those in the outermost orbit around the atom; these are the atom's valence electrons. In the case of conductors, the amount of energy required to remove an electron from its orbit is very small. For example, the heat present at room temperature is sufficient to detach an average of one electron per atom of copper; the detached electrons are then free to move under the presence of an electric field. On the other hand, the atoms of an insulator must see a great deal of energy before they will give up electrons. Their valence orbitals are nearly or completely full and very stable; those of the conductors are conversely nearly empty, and hence the electrons present are relatively loosely attached. Semiconductors are at the midway point; their valence electrons occupy half of the available positions. The two most common examples are silicon and germanium; each has four valence electrons with room for eight. The desire to fill the valence orbit is so strong that atoms of silicon or germanium will actually share electrons as shown in Figure 2.6. The bond between two such atoms is called a covalent bond. An atom of silicon shares each of its four valence electrons with four other atoms, all of which are three-dimensionally equidistant from one another. (Figure 2.6 and those that follow are two-dimensional for the sake of clarity.) In this fashion, each atom sees eight valence electrons and is satisfied; atoms arranged in such a regular format form a crystal. At a temperature of absolute zero, all of the valence electrons are fixed in place and no current can flow as there are no free electrons.

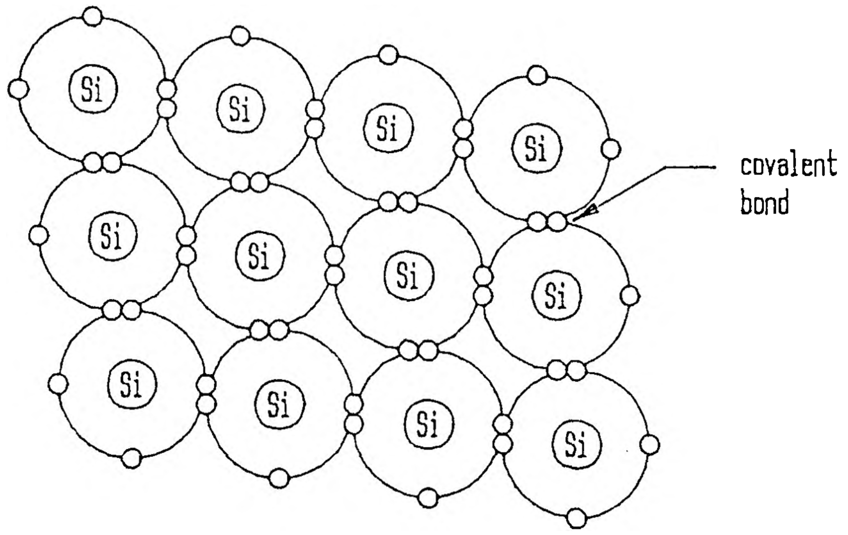


Figure 2.6. Covalent Bonding of Semiconductor Atoms

When the ambient temperature of a semiconductor crystal rises, the heat energy applied to the electrons will lead to an occasional breaking of a covalent bond. When this happens, the electron that has left its valence position is free to move and carry current; the position vacated by it is termed a hole. Other electrons in the crystal will be attracted to take the position of the hole. When this occurs, the hole itself is said to move from its current position to that of the attracted electron. In any pure amount of a semiconductor, there will be an equal number of holes and free electrons, as one hole is created for each electron that leaves its bond; such a pure semiconductor is said to be intrinsic. The holes and electrons are in a constant state of generation and recombination. The generation and recombination rates of electrons and holes are equal and constant at a given temperature, as dictated by the equilibrium of the material.

b. Extrinsic Semiconductors. As described previously, intrinsic semiconductors conduct current only poorly as the valence electrons present form exactly the number of covalent bonds necessary to satisfy each atom's desire for a full outermost orbital. This situation may be disrupted by adding atoms of another element with a different valence configuration to the original pure semiconductor. This process is known as doping; the resulting semiconductor is said to be extrinsic, or no longer pure. Consider as an example the element phosphorous. It has five valence electrons, one more than the four of silicon. Figure 2.7 shows the resulting crystalline structure when a phosphorous atom is present amidst silicon atoms. Four of the five phosphorous valence electrons form covalent bonds with the adjacent electrons. The remaining phosphorous electron will require very little energy to be freed from its orbit; the heat of room temperature is sufficient for this purpose. This electron is thus readily available for carrying current. The original phosphorous atom is referred to as a donor atom as it donates one electron for carrying current through the semiconductor. The phosphorous atom becomes a positive ion when it gives up the extra electron, as there is now a proton present which is not negated by the absent electron. A ratio of one such electron to every one hundred million atoms of the semiconductor germanium results in an increase in conductivity by a factor of sixteen; germanium doped in such a fashion is useful for transistor applications.

Semiconductors doped by the addition of extra electrons are categorized as n-type semiconductors. The n refers to the negative charge of the extra electrons present. In the presence of a voltage difference, these electrons will be drawn toward the positive voltage contact, at which point they will leave the semiconductor; they are simultaneously replaced by new electrons entering the semiconductor at the negative battery contact. Note that at no time does the semiconductor itself actually maintain any charge, positive or negative; all electron charges are offset by those of

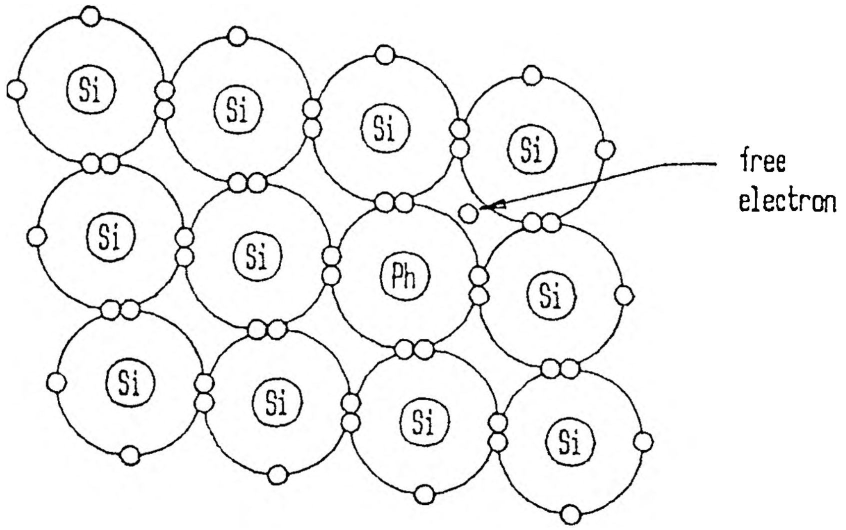


Figure 2.7. n-type Semiconductor Crystal Structure

corresponding protons present in the phosphorous ions, and a constant number of electrons is present in the semiconductor at all times. It should also be observed that the charges associated with the positive ions are fixed in the geometric structure; the negative charges of the donated electrons are capable of movement through the semiconductor.

Doping may also be performed to shift the number of electrons present for covalent bonding in the reverse direction. Consider the element aluminum with its three valence electrons. When added to a semiconductor such as germanium, the configuration shown in Figure 2.8 results. Observe that the three aluminum valence electrons form covalent bonds with three of the four neighboring germanium atoms. The fourth germanium atom cannot form a covalent bond, although it would strongly like to. The unoccupied position of this desired electron is a hole, as described for intrinsic semiconductors. The difference is that this hole and others like it do not have corresponding electrons present in the extrinsic semiconductor. In order to fill the hole and complete the electron pair and hence the orbital, an electron from a nearby germanium atom will be attracted to this position. Correspondingly, the hole itself moves to the now unoccupied position. The original aluminum atom is said to be an acceptor atom as its deficiency in valence electrons allows for the acceptance of an extra electron. When this occurs, the accepting atom becomes a negative ion, as the charge on the added electron is not countered by a proton. An effective positive charge is present at the position the hole occupies as there is one proton which is no longer balanced by the now-missing electron.

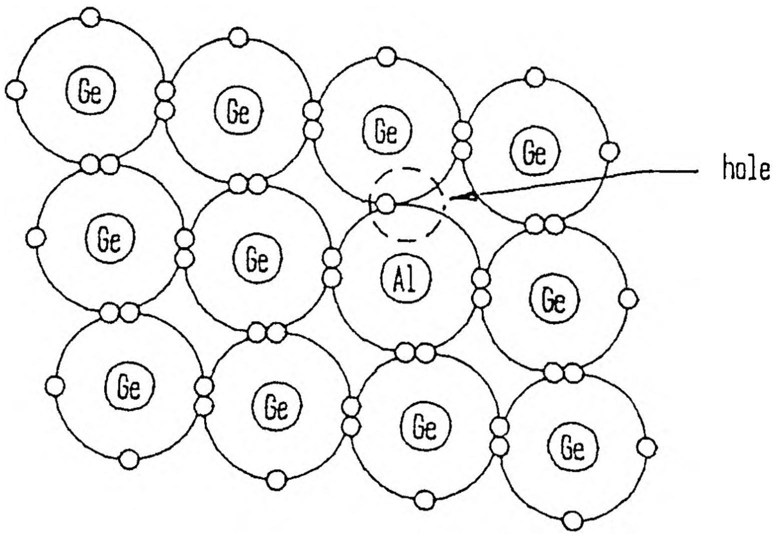


Figure 2.8. p-type Semiconductor Crystal Structure

When acceptor atoms are added to a semiconductor, the doped result is referred to as a p-type semiconductor. The p here refers to the positive charge of the holes resulting from the mixture. When a voltage is applied across a p-type semiconductor, the positive holes will be drawn toward the negative terminal. As a hole reaches this position, an electron leaves the terminal and enters the semiconductor, occupying the empty hole. Simultaneously, an electron leaves the semiconductor and enters the positive terminal, creating a new hole. This hole then begins to move along with all of the other holes present in the semiconductor toward the negative terminal to continue the process. As with the n-type semiconductor, the p-type semiconductor as a whole does not ever become charged; the positive charges of the holes are balanced by the negative acceptor ions, and the number of holes present in the semiconductor remains constant. It is also important to note that in contrast to the n-type semiconductor, it is the negative charges located at the acceptor atoms which remain in a fixed position while the positive charges associated with the holes move throughout the semiconductor material.

Within an extrinsic semiconductor, there is an excess of one type of current carrier present; it is called the majority carrier. For n-type semiconductors, it is the electron and for the p-type it is the hole. The energy due to the temperature to which the semiconductor is exposed will lead to the introduction of current carriers besides the majority carriers due to doping. Thus holes will be present in an n-type semiconductor, although they will be far fewer in number than the free electrons. Likewise, some electrons will be present for the conduction of current in p-type semiconductors. Holes in n-type semiconductors and electrons in the p-type are referred to as minority carriers; they flow in opposition to the majority carrier present. It is important to note here that if a number of minority carriers are introduced in a semiconductor, they will quickly disappear upon combination with majority type carriers; electrons disappear into holes, and holes are filled by electrons.

c. The p-n Junction. Consider what happens when a p-type semiconductor is joined with an n-type semiconductor as in Figure 2.9. From the n-type semiconductor, free electrons near the junction will begin to move across it and occupy the nearby holes in the p-type semiconductor. As they do so, the positive ions with which they were associated are left without balancing negative charges, and a positive charge is built up in the n-type semiconductor. Correspondingly, as the electrons take up positions in the p-type semiconductor, they build up a negative charge there for which there is no balancing positive charge. From the p-type semiconductor, holes near the junction will move into the n-type semiconductor by attracting nearby electrons across the junction to take their positions. As they do so, they leave behind their charge counterparts, negative ions. This also increments the negative charge in the p-type semiconductor. As the holes move across the junction, the positive charges associated with them, without any balancing negative charges, increment the positive charge of the n-type semiconductor. A point is rapidly reached at which the negative charge built up in the p-type semiconductor near the junction is sufficient to prevent any more electrons from crossing the junction. Likewise, holes will be unable to cross as they will be repelled by the positive charge in the n-type side near the junction. The regions of the two sides which have given up their current carriers is termed the depletion layer. The junction between the two types of semiconductor is now referred to as the junction barrier, as it opposes the movement of further majority current carriers by repelling them.

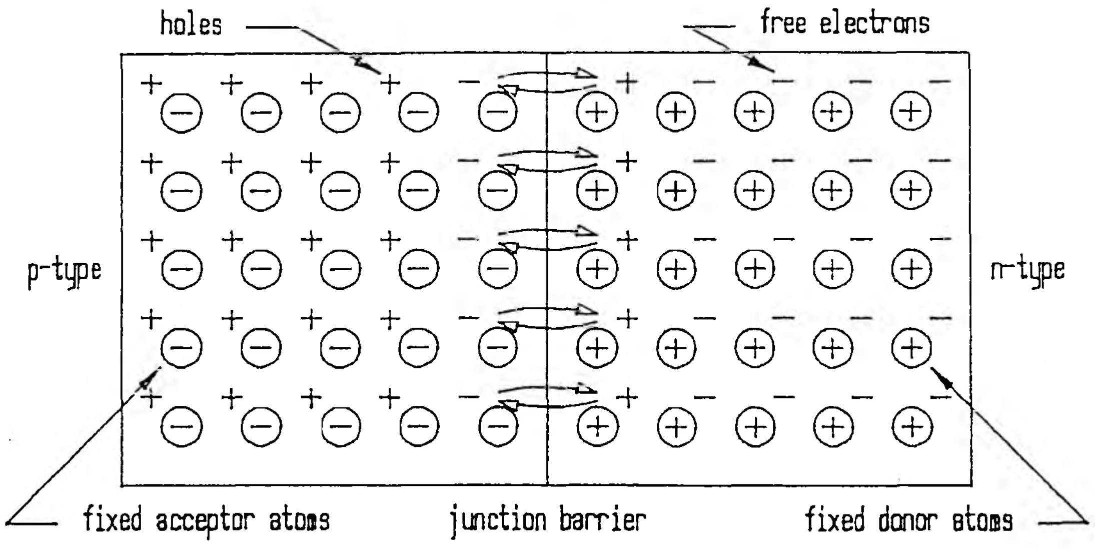


Figure 2.9. p-n Junction of Semiconductors

When a voltage difference was placed across a pure semiconductor of either type, current was seen to flow without regard to direction. This is not the case for the p-n junction of semiconductors. Consider first what occurs when a positive contact is placed at the p-type end of the junction and the negative contact at the n-type, as in Figure 2.10. When the voltage difference is increased from zero, the holes in the p-type end are repelled from the positive contact toward the also-positive end of the junction barrier; the free electrons in the n-type end are likewise repelled from the negative contact toward the negative end of the junction barrier. As holes are pushed toward and then into the depletion layer, the negative charge present is somewhat diminished. Likewise, electrons repelled by the negative contact lessen the positive charge of the n-type end of the depletion layer. As holes and electrons are again capable of being near the semiconductor junction as described for the initial merge, some of each will have sufficient energy again to cross the junction to the other side. When this happens to an electron, it will shortly recombine with one of the many holes present in the p-type side. Similarly, when a hole crosses the junction it quickly recombines with one of the many free electrons present in the n-type side. When a hole disappears, it is replaced by the break up of a covalent bond near the positive contact. The new hole is repelled by the positive voltage toward the junction; the electron is drawn to the positive terminal and enters it, while another electron simultaneously exits the negative terminal to enter the n-type semiconductor. It is also repelled toward the junction.

At the same time, a small number of minority carriers continue to form as energy is applied to the semiconductor by way of ambient temperature. When a free electron and hole appear in the p-type end, the electron is immediately attracted toward the connecting positive terminal. Likewise, when a free electron and hole appear in the n-type end, the hole is drawn to the nearby negative terminal. An electron will leave this terminal to occupy the hole while some free electron drawn to the positive terminal simultaneously leaves the p-type end of the semiconductor to enter the adjacent terminal. Thus a small current will flow through the junction of semiconductors by minority carriers.

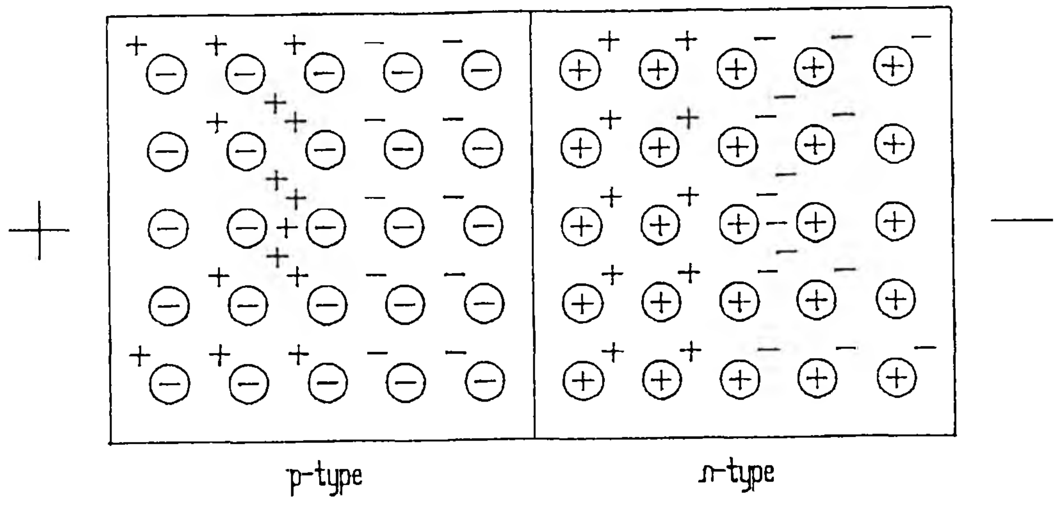


Figure 2.10. Forward Voltage Applied to a p-n Junction

As the voltage across the p-n junction of semiconductors continues to be increased, a level is reached at which the charges present in the depletion layer are overcome and current flows freely. For example, this level is around 0.6 volts for silicon. Further increase in voltage rapidly increases the current flow. When a p-n junction has a positive voltage applied to the p-type end and a negative to the n-type end as described here, the voltage is said to be a forward voltage, or bias. The current due to the minority carriers is not influenced by changes in voltage; it forms only a negligible portion of current under a forward bias.

A reverse voltage, or reverse bias, is applied to a p-n junction of semiconductors when the positive terminal is attached to the n-type end and the negative to the p-type end, as in Figure 2.11. Under these circumstances, the free electrons of the n-type semiconductor will be drawn away from the junction toward the positive contact. Likewise, the holes of the p-type end will be drawn away from their end of the junction and toward the negative contact. The positive and negative ions resulting from the electrons and holes, respectively, that are attracted away combine their charges with those of the ions already alone around the p-n junction. The result is a widening of the depletion layer, and the flow of current by majority carriers is further resisted.

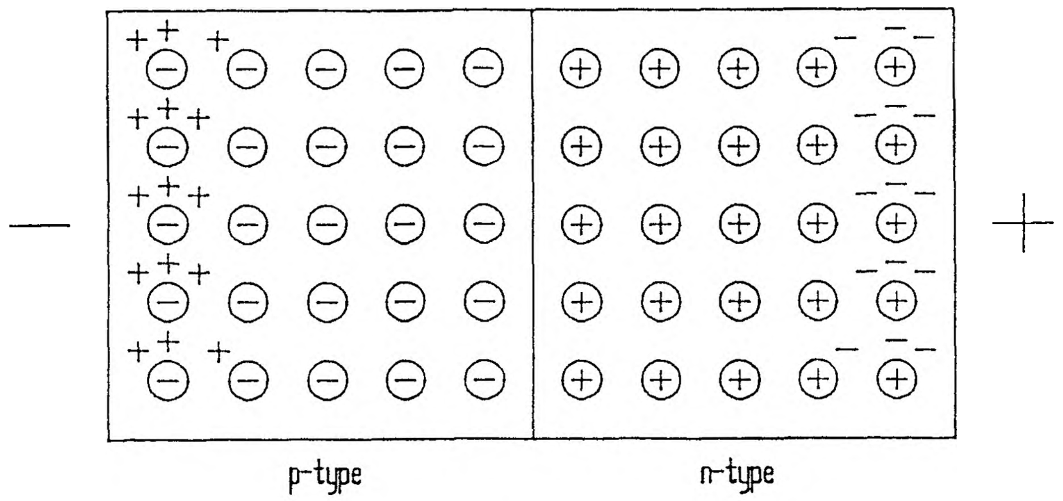


Figure 2.11. Reverse Voltage Applied to a p-n Junction

Minority carriers, on the other hand, will continue to form as energy is applied in the form of heat. When an electron is freed in the p-type semiconductor, it is attracted across the junction toward the positive terminal. Similarly, holes appearing in the n-type semiconductor are pulled across the junction toward the negative terminal. Electrons will leave the negative terminal to fill the holes while electrons drawn to the positive terminal simultaneously exit the junction of semiconductors. Thus a small current flows by way of minority carriers. The creation of holes and free electrons occurs as energy is applied to the semiconductor in the form of ambient heat, as described before; at any fixed temperature, the rate of generations for holes and electrons is also fixed. Increasing the voltage present at the terminals then has no immediate effect on the amount of current flowing as all minority carriers move upon creation. It is important to note however that as the voltage increases, the pull on the minority carriers also increases, causing them to move faster and faster through the semiconductors. There is a voltage at which the minority carriers will be moving so quickly that they will cause some covalent bonds to break, producing more carriers and, consequently, an increase in current. These new carriers will add to this breakdown and increase in current. The voltage around which this phenomenon begins to occur is referred to as the reverse breakdown region.

d. The Diode. The p-n junction, unlike its single component semiconductors, has been seen to behave differently under different conditions: a sizable current can flow through the p-n junction when a forward bias is applied (positive contact to the p-type semiconductor, negative contact to the n-type), but no appreciable current will flow under a reverse bias. These characteristics are extremely useful in a wide range of applications in electronics; the junction of a p-type and an n-type semiconductor is commonly referred to as a diode. The circuit symbol for the diode is given in Figure 2.12 (a). Note that the arrow head indicates the direction current may flow; this is opposite to the direction of electron movement because of the original conception that current flowed from the positive to the negative. The light-emitting diode, or LED, included in the power supply circuits earlier, functions just as described here. The particular semiconductors it is formed from produce photons in the visible spectrum when electrons and holes recombine; the LED is consequently illuminated only when a forward bias causes current to flow and many recombinations are occurring. Other diodes also give off light energy, but it is not visible to the eye. The circuit symbol for the LED is given in Figure 2.12 (b). The arrows are used to indicate the light-emitting property.

e. The pnp Transistor. The transistor, the device to be used here for switching, utilizes the properties described for the diode; however, in addition to the two semiconductor layers of the diode, the transistor has a third. The layers of a transistor may be either p-type, n-type, and p-type again (pnp) or n, p, and n (npn). For this

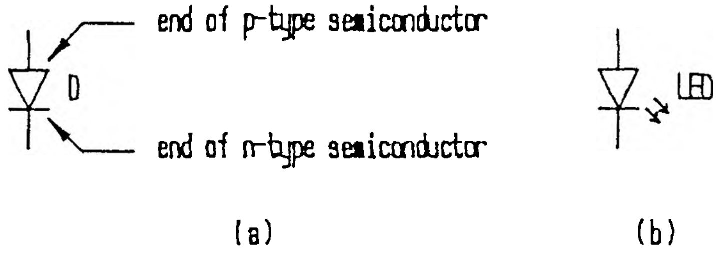
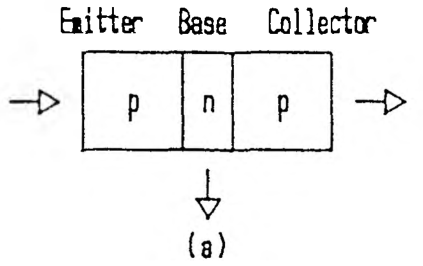


Figure 2.12. Diode Circuit Symbols

project the pnp transistor has been selected and will be discussed here; the characteristics of the pnp and npn transistors are different but the concepts are the same. In particular, the transistor to be discussed is of the grown junction variety; the point-contact transistor, another type, is mechanically different.

The pnp transistor is depicted in Figure 2.13 (a), along with the designations of the three layers; the first p-type region is the emitter, the second p-type region is the collector, and the intervening region, or base, consists of n-type semiconductor. It is worth mentioning here that while the construction appears to be symmetric, the p-type regions are not interchangeable and the transistor must only be used in the specified orientation for reasons which will be made clear later. The transistor has two states; in the first, current flows through the transistor from emitter to collector and in the second no current flows through the transistor at all. The arrows in Figure 2.13 (a) indicate the direction of current flow through the semiconductor regions. The voltage condition present at the base controls the presence or absence of current through the transistor. Figure 2.13 (b) shows the circuit symbol for the pnp transistor. The arrow here indicates the emission of holes into the base.



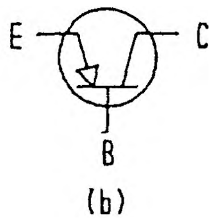


Figure 2.13. The pnp Transistor

Consider first the conditions that are necessary to permit the flow of current. To begin with, the p-n junction between the emitter and base must be under a forward voltage bias. As described for the diode earlier, this will lead the holes in the emitter region to cross the junction barrier into the base, while free electrons in the base will cross the barrier into the emitter; as with the diode, the forward voltage must be above a certain level to bring about the current flow. The first p-type region is designated the emitter as it emits holes into the base. Also as in a diode, the holes coming into the n-type region would quickly recombine with the free electrons found there. The second junction, an n-p transition between the base and the collector, must have a reverse bias applied to it, making the collector more negative than the base. Thus when holes enter the base, and become minority carriers in the n-type region, they are immediately affected by the pull of the more negative terminal at the end of the collector p-type region and do not all recombine with free electrons. Holes do not enter the base from the collector due to the reverse bias. The second p-type region is designated the collector as it collects holes from the base. Note that the holes appear to have been thermally generated to the base-collector junction of semiconductors. Some of the holes will recombine in the base, but many of them will cross the second junction barrier as minority carriers and then move through the collector to the connecting negative terminal due to the construction of the transistor layers. Upon the hole's arrival at the negative terminal, an electron leaves the terminal and occupies the hole. Simultaneously, a covalent bond in the emitter near the positive terminal breaks down, and the electron freed exits the emitter for the positive terminal. The hole just created then begins moving toward the emitter-base junction to continue the process. When a hole does recombine with a free electron in the base, another electron enters the base from the adjoining terminal to take its place. This small base terminal input of electrons combines with the much larger input from the collector's terminal to form a flow equal in magnitude to the flow of holes into the base from the emitter. The two conditions required for current flow may be met simultaneously be placing a voltage at the emitter p-type semiconductor which is greater than that at the n-type base, which must in turn be greater than the voltage present at the p-type collector.

The object in the construction of a transistor is to make the conditions present in the base as favorable as possible for permitting holes injected from the emitter to travel on through the base into the collector region; if the holes simply exit through the base nothing is achieved. This enhancing of conditions is accomplished in different ways. One aspect of transistor construction calls for the width of the base to be very narrow, typically a few ten-thousandths of an inch or less. With a relatively small distance to travel in the base's n-type region, few holes have a chance to recombine with the free electrons present there. The density of electrons present in the base is also reduced in comparison to the hole density of the emitter to further lessen the chance of encounters between injected holes and free electrons. The hole densities in the two p-type regions are also adjusted differently in each to achieve optimum results. For this and other reasons a transistor cannot have its emitter and collector interchanged, even though the simple depiction in Figure 2.13 makes the transistor look symmetric. The efficiency of the transistor is based on the percentage of holes that continue into the collector without recombining in the base. Percentages ranging from 95% to 99% are readily achievable.

In order to prevent current from flowing through the transistor, the emitter-base junction's forward bias must be eliminated. This can be accomplished by raising the voltage at the base to the point where electrons and holes can no longer easily cross the emitter-base junction barrier. Thus the flow of current through the transistor from emitter to base is controlled by the voltage at the base: when high, no current flows and when low, a forward bias is in effect allowing the junction barrier to be crossed and current to flow.

The transistor has been around since the late 1940's, and much more remains to be said about precise operating characteristics, different versions of the transistor, and so on. The material included here is for the purpose of acquainting the reader with transistor technology; further detail may be found in a number of works, including [Hibb65], [Casa73], [Krug54], [Walk66], [Bedf64], and [Malm69].

For this particular problem, the 2N2907A pnp transistor is employed. Two of these transistors are used for the circuit of the motor being discussed. Recall that the motor had one lead connected to the +4.5 volt source; the other unused end could be connected to either ground or the +9 volt source to achieve bi-directional rotation. The first transistor is positioned with its emitter at the unused motor lead and with its collector going to ground; see transistor Q1 in Figure 2.14. The second transistor is placed with its emitter at the +9 volt source and with its collector going to the same motor line; this is transistor Q2. Consider now transistor Q1. The +4.5 volt source on the other side of the motor provides one end of the potential needed for the forward biasing of the emitter-base junction; bringing the base low puts the forward bias in effect and current flows from emitter to collector, or from the motor line to ground.

The circuit is completed and the motor begins to turn as expected. Note there is actually somewhat less than +4.5 volts across the motor as a small voltage drop exists across the transistor.

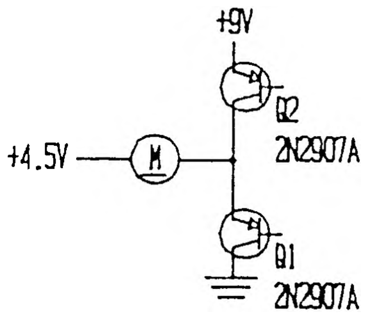


Figure 2.14. The Motor Sub-Circuit

The second transistor, Q2, is in a different situation, however. Its emitter is at the +9 volt potential and the other side of the motor is at the +4.5 volt level, not the ground. When the base is brought low directly, a forward bias is in effect across the emitter-base junction as desired, but there is also a forward bias across the base-collector junction, as the base is at ground while the collector sees +4.5 volts on the other side of the motor. Thus current will flow from the emitter and out the base and also from the collector and out the base. Under these circumstances the motor turns in the same direction as it did when the base of transistor Q1 was made low. This situation will be remedied by the introduction of another circuit element, the resistor.

3. Resistors. To prevent the reverse current flow described above, a resistor will be placed before the base of the second transistor. The resistor is a passive device, as is its function. Conductors such as copper allow the flow of current, or movement of electrons, to take place freely. Resistors hinder the movement of electrons by employing materials other than conductors for bearing current. One technique for the construction of resistors is to combine carbon with an electrically-inert filler. As the

amount of carbon is increased with respect to the filler, it becomes easier for current to flow through the combination. As current is not allowed to flow freely from one end of the resistor to the other, a voltage drop will occur across the resistor.

The resistor voltage drop noted above can be used to achieve the proper biasing of the second transistor in the motor control circuit. In fact, resistors are placed before both of the circuits' transistors as seen in Figure 2.15. (Recall that resistors R1, R2, and R3 were used in the power supply circuits.) The resistor R5 at the base of transistor Q2 is of primary concern. The voltage at the transistor base must be between that at the emitter and collector for the emitter-base forward bias and base-collector reverse bias to be in effect. The voltage drop across a resistor is given by Ohm's Law, which states that the voltage drop is equal to the product of the current passing through the resistor and the amount of resistance (V = IR).

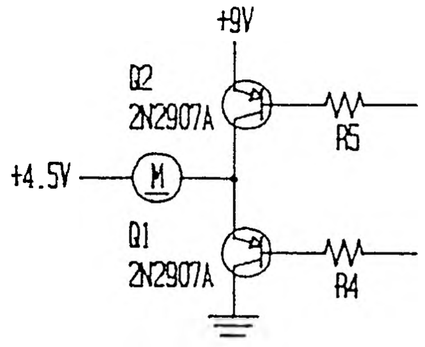


Figure 2.15. The Modified Motor Sub-Circuit

Experimentation was carried out for each of the six motor control circuits as different joints were found to operate better with varying amounts of current exiting through the transistor base. Table 2.1 states the resistances used for the various motor control circuits. Note that the values vary across a relatively small range, 100 to 470 ohms. The resistances placed at the bases of the transistors whose emitter and collector lie at +4.5 volts and the ground, respectively, were not absolutely necessary.

However, including a small resistance guarantees that the holes entering the base from the emitter will see a more negative pull from the collector than from the base terminal. Resistances of 10 ohms were used here.

Table 2.1. RESISTANCES USED AT BASE OF SECOND TRANSISTOR

Transistor	Joint	Resistance (ohms)
Q2	Wrist Up	470
Q4	Gripper Open	100
Q6	Elbow Left	220
Q8	Arm Up	470
Q10	Wrist-Rotate Right	470
Q12	Arm Right	150

4. The Computer. To this point, the electronic interface consists of twelve transistors and twelve corresponding resistors. As described, these elements are on the layout's motors-and-robot side. Consider now the computer side of the arrangement. For the twelve transistors and their corresponding lines that are to be raised and lowered, the computer used for control must provide the outputs for these twelve lines. If a computer possessing twelve output lines were used, connections could possibly be made directly to the computer's output port; fewer lines will require additional hardware. For this project, the IBM Personal Computer was selected. The data lines of a printer port are suitable for the required robot control. The IBM PC has the advantage of being in widespread use and is thus generally accessible. The drawback for this project is that the printer port has only eight data lines; this problem is easily surmountable, though. In addition, by working with just the eight data lines the project is also transferrable among a wide number of computers.

5. The Decoder. In order to achieve the transformation from the eight available output lines to the twelve required, a decoder will be used. A decoder is a device which

accepts as its input a number of lines whose voltages are effectively interpreted as bit positions in a binary number; for the device to be used, high voltages (+5 volts) will be regarded as ones and low voltages (0 volts, or ground) as zeroes. Thus if a decoder having three inputs sees, in order, high, low, and again high voltages, it will interpret this as a binary 101, or decimal 5. The decoder then takes action by raising or lowering the voltage on the one of its output lines designated by the binary number on the inputs; for the device to be used, the selected output is made low (again, 0 volts) while all others remain high (again, +5 volts). Of course, the number of output lines the decoder has is dependent upon the number of inputs. For example, for two inputs there are 2^2 or four possible interpretations, 0, 1, 2, and 3; three inputs can generate any of 2^3 or eight possible interpretations, 0, 1, 2, 3, 4, 5, 6, and 7; four inputs require 2^4 or sixteen output lines, and so on.

a. The 1-to-2 Decoder. To see how the decoder works, consider as an example a 1-to-2 decoder using the same voltage representations as those of the decoder to be used; this discussion is based on that of Lewin [Lewi83]. This device would see as its input a single line which would be at either a high or low voltage. Two output lines would be required. The first of these would be the zero output; it would be low when the input is zero, or low, and high when the input is one, or high. The second output line would be the one output; it would be high when the input is zero, or low, and low when the input is one, or high. Table 2.II organizes these requirements.

Table 2.II. OUTPUTS FOR 1-TO-2 DECODER

Input Line	Output 0	Output 1
L	L	H
H	H	L

It is clear from the table that the states of the zero output line are identically those of the input line while the one output states are the opposite of the input line states. The one output line is said to be inverted from the original; to obtain the states required for the one output, a device called an inverter is employed. The inverter sees a single input line, at either a low or high voltage, and produces a single output, whose state is opposite that of the input, high or low, respectively. Thus before the decoder can be fully explained, the inverter itself must first be examined.

b. The Inverter. The transistor was introduced in a preceding section as its capability to act as a switch was of use for controlling the motors of the robot. The transistor is useful again here in that it provides the basis for the inverter. Consider Figure 2.16 (a). The pnp transistor Q is situated so that its base is connected to the voltage to be inverted; resistor R1 is selected so as to place the base current at an appropriate level. The emitter is tied directly to a line supplying a high voltage. The collector provides the output of the inverter, but it is also connected through resistor R2 to a line supplying a low voltage, or ground. When the input line is high, the base of the pnp transistor is not provided with the low voltage necessary for the forward biasing of the emitter-base junction and the reverse biasing of the base-collector junction; consequently, no current flows and the output line is at the low voltage level through resistor R2. When the input line is brought low, the proper biases are in effect and current flows from emitter to collector. Current leaving the collector follows the output path, as resistor R2 deters electron movement along the other path, and the output line is at a high voltage through the transistor. In this fashion an inverted output is obtained from a given input. It should be noted that this is but one way to accomplish the desired effect; one alternative employs the npn transistor, mentioned earlier, and a corresponding reversal of voltages. The circuit diagram for the inverter is shown in Figure 2.16 (b). It is worth noting here that the inverter performs the unary logic operation NOT on its input.

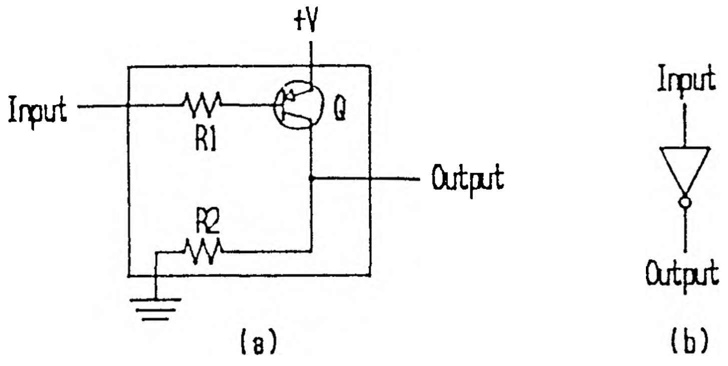


Figure 2.16. The Inverter

With the inverter in hand, the 1-to-2 decoder follows quickly. Figure 2.17 shows the decoder consists of nothing more than splitting the input line and running one of the two resultants through a single inverter. The unaltered line is the zero output, and the inverted line is the one output. Note that one of the two lines will always be high and the other always low; if this decoder were to be used with two of the motor-controlling transistors, the motor would always be turning in one direction or the other.

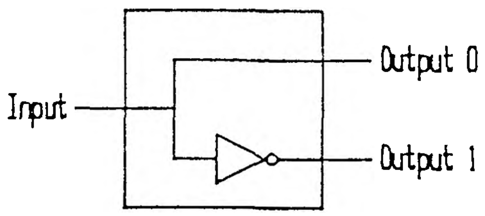


Figure 2.17. The 1-to-2 Decoder

c. The 2-to-4 Decoder. A more practical and consequently more complicated example can be found in the 2-to-4 decoder. There are now two input lines to contend with, each of which may be at either a high or low voltage. There are then four possible combinations of inputs and thus four output lines. Table 2.III states the desired outputs for the possible input combinations.

Table 2.III. OUTPUTS FOR 2-TO-4 DECODER

Input Lines		Output Lines			
1	0	3	2	1	0
L	L	H	H	H	L
L	H	H	H	L	H
H	L	H	L	H	H
H	H	L	H	H	H

From the table, it is clear that the zero output line is low only when both of the inputs are low; similarly, the one output is low when the zero input is high and the one input is low, and so on. Recalling the use of the inverter in the 1-to-2 decoder, the requirements for low voltages on the output lines may alternatively be stated in terms of the presence of high voltages on the input lines. For example, the zero output line is low if and only if input line 0 is not high and input line 1 is not high; the one output line is low if and only if input line 0 is high and input line 1 is not high, and so on. The inverter can be used to raise low inputs so that an output line is to be lowered only in the presence of two high voltages. Another electronic device, the AND gate, will make this detection possible.

d. The AND Gate. The AND gate has a single output which is high if and only if its two inputs are both high as well. Like the inverter, the AND gate is a semiconductor product. The desired function may be obtained in a number of different ways. Figure 2.18 (a) shows one of the simplest configurations, requiring only two diodes and a resistor. When the voltage of input 0 is low, diode D1 is forward biased by way of resistor R to a positive voltage; thus current flows through the diode, leaving the output at the same level as input 0, or low. Similarly, when the voltage of input 1 is low, diode D2 is forward biased, and the output is at the voltage level of input 1, or low. Certainly when inputs 0 and 1 are both low the output is also low as current is capable of flowing through both diodes. In the single case where inputs 0 and 1 are both high, diodes D1 and D2 are both reverse biased; no current flows through either diode, so the output is at the high voltage level by way of resistor R. In this manner the requirements for the logical AND operation are met. Figure 2.18 (b) shows the circuit symbol for the AND gate.

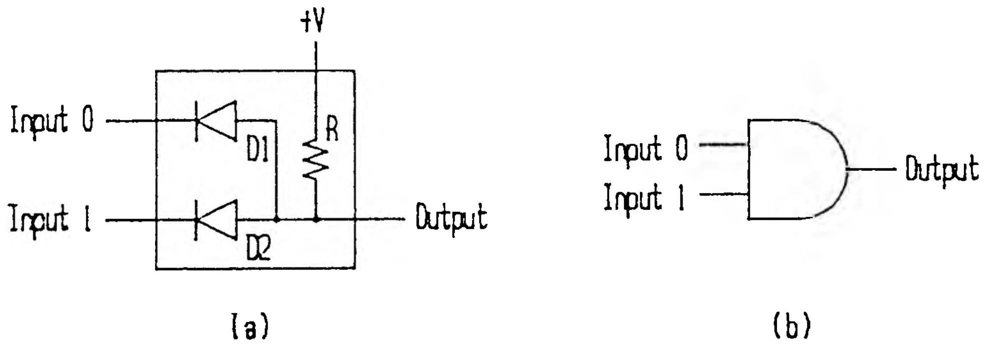


Figure 2.18. Diode-based AND Gate

Another alternative for the AND operation involves the use of transistors. Figure 2.19 demonstrates an example of how the desired pairings of inputs and outputs may be achieved. Input 0 is fed to the base of pnp transistor Q1 through resistor R1, whose resistance determines the base current for Q1. Likewise, input 1 is fed to the base of pnp transistor Q2 through resistor R2. When input 0 is low, transistor Q1 conducts current from emitter to collector, putting the output at the same voltage level as the collector, which goes to ground. A low input 1 causes transistor Q2 to conduct, putting the output at its collector voltage level, or zero. Certainly when both of the inputs are low, the output will again be at the common collector level, zero. When both inputs are high, however, neither transistor conducts current, and the output is at the high voltage level through resistor R3. There are, of course, many other implementations that achieve the same result.

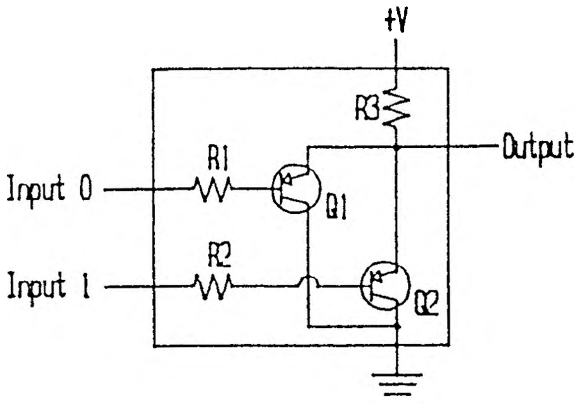


Figure 2.19. Transistor-Based AND Gate

With both the inverter and the AND gate available, the 2-to-4 decoder may be constructed as shown in Figure 2.20. The connections made can be reconciled with Table 2.III, stated earlier. Output 0 is low when both of the inputs are low, so the AND gate associated with it receives inputs from the inverters of each of the two decoder inputs. Thus when both inputs are low, the inverted lines are high, and the AND gate produces a high voltage. This is in turn inverted to produce the desired low voltage for Output 0. Output 1 is low when input 1 is low and input 0 is high, so its AND gate receives as input the inverted voltage from input 1 and the original voltage of input 0. When input 0 is high and the inverted voltage of input 1 is high as well (meaning input 1 is itself low), the AND gate yields a corresponding high voltage, which is also inverted to obtain output 1. Outputs 2 and 3 are generated in similar fashion. (The AND gate and inverter are commonly combined as a single circuit element, the NAND gate, whose output is low only when both of its inputs are high.)

e. The 4-to-16 Decoder. Returning now to the original problem, it was required to take the eight available control lines originating from the computer and lower one of the twelve control lines leading to the transistors. As has been demonstrated, the decoder effectively functions by interpreting its input as a binary number and placing

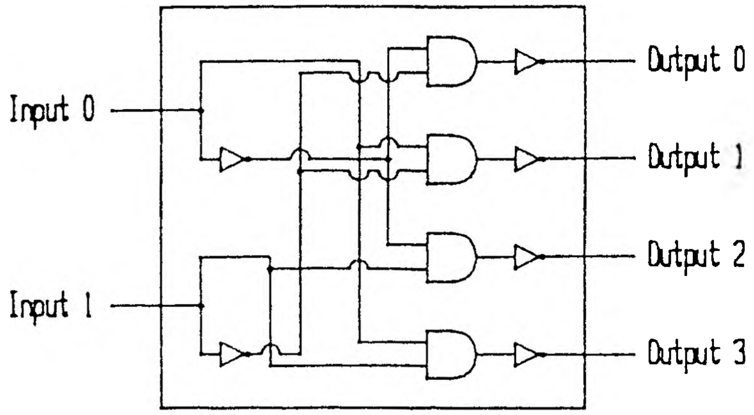


Figure 2.20. The 2-to-4 Decoder

only the correspondingly-numbered output line at a low voltage. One input may select either of 2^1 or 2 outputs; two inputs may select from among 2^2 or 4 outputs; three inputs produce 2^3 or 8 possible outputs, and so on. The particular problem requires 12 lines for controlling the motor transistors; since 12 is larger than 8 but less than 2^4 or 16, a 4-to-16 decoder will be adequate for the task. Note also that while twelve of the lines will be directly used, a thirteenth will be made high when all of the twelve lines are to be low simultaneously; this corresponds to the intervals during which no motor is running in either direction.

Consider how such a device might be constructed. Extrapolating from the 2-to-4 decoder, it is clear that each of the sixteen decoder outputs is determined by a unique combination of the voltages on the four input lines. Table 2.IV shows the specific pairings.

As was done for the 2-to-4 decoder, each of the 4-to-16 decoder's output lines is the inverted output of an AND gate. The difference here is that while the 2-to-4 decoder AND gates accepted two inputs, each of the AND gates in the 4-to-16 decoder must be able to accept four inputs. The two example constructions for the AND gate

Table 2.IV. OUTPUTS FOR 4-TO-16 DECODER

Input Lines 3	2	1	0	Output Lines 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
L	L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L
L	L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H
L	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H
L	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H
L	H	L	L	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H
L	H	L	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H
L	H	H	L	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H
H	L	L	L	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H
H	L	L	H	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H
H	L	H	L	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H
H	L	H	H	H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H
H	H	L	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H
H	H	L	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	L	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H
H	H	H	H	L	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H

given in Figures 2.18 and 2.19 may be easily modified to accept four inputs; the design of Figure 2.21 shows how this is readily accomplished by expanding the arrangements in the simpler versions.

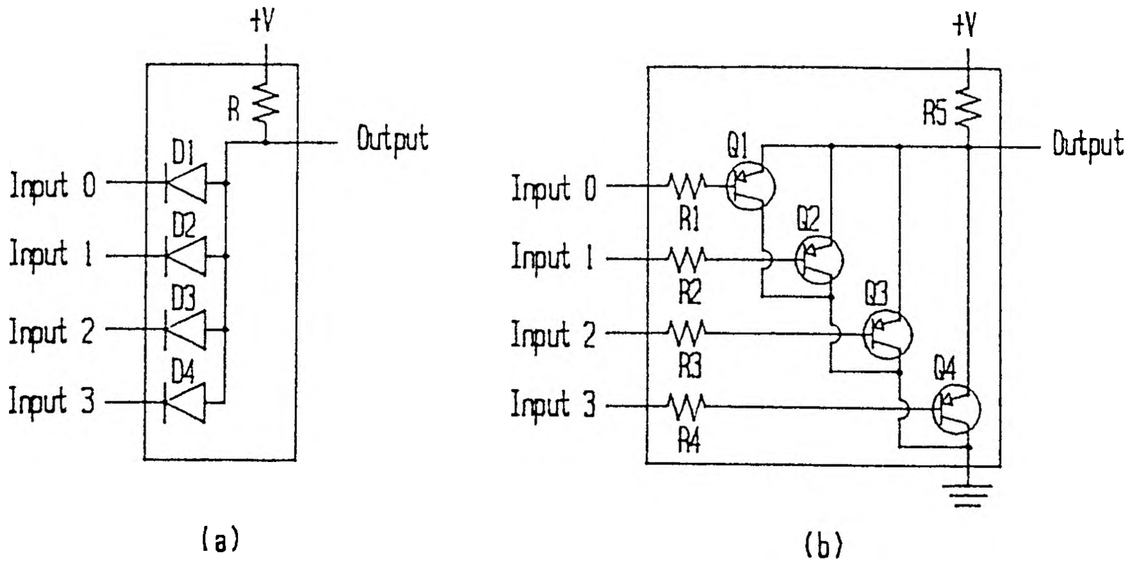


Figure 2.21. 4-Input AND Gates

As with the 2-to-4 decoder, each input will be split, and one of the two resulting lines will be fed to an inverter so that the AND gates can produce a high output for a mixture of originally high and low inputs. Figure 2.22 shows the complete construction. Note that there are twenty-four inverters and sixteen 4-input AND gates employed. As seen in Figure 2.21, each of the AND gates will require either four diodes or four transistors, as opposed to the two used in the two-input AND gates. Consequently, the cost of the 4-input AND gates will be higher while the reliability is reduced by the increase in the number of parts.

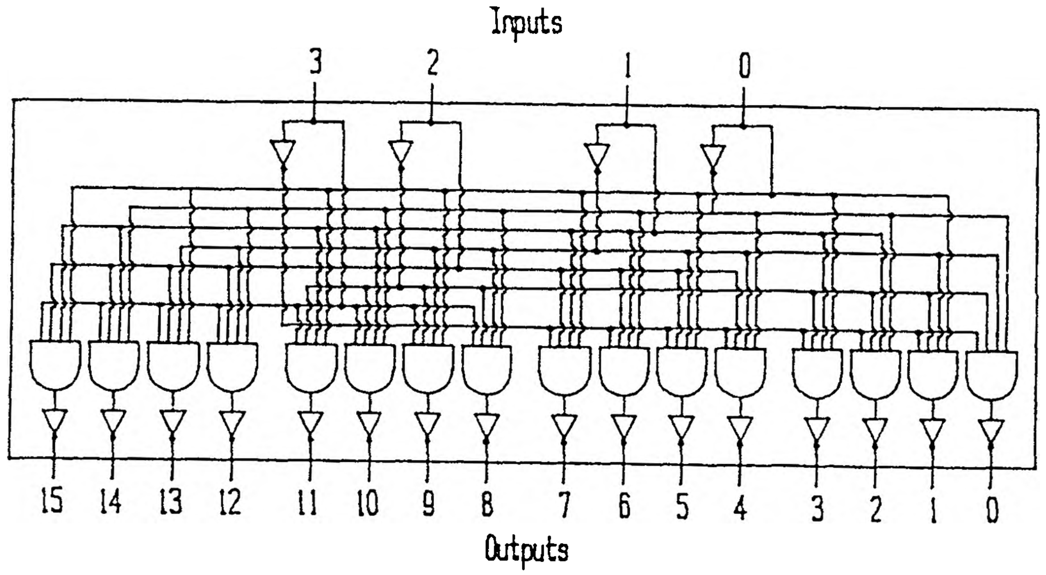


Figure 2.22. The 4-to-16 Decoder

An alternative to the decoder requiring 4-input AND gates can be realized by employing as an element within the 4-to-16 decoder the 2-to-4 decoder developed previously. The four control lines of a 2-to-4 decoder may be used to select from among four alternatives. If the sixteen required output lines of the 4-to-16 decoder are divided into four groups of four, the different groups may be thought of as being selected one at a time by the output lines of the 2-to-4 decoder. This is accomplished by having each of the sixteen 4-to-16 decoder outputs be the inverted output of an AND gate, the input for which in each case will include the aforementioned selection line after inversion; see Figure 2.23. Thus any one of the sixteen decoder outputs can only be low when its AND gate input from the 2-to-4 decoder and inverter is high; conversely, as a single 2-to-4 decoder output will be low at any one time, there are only four possible outputs of the 4-to-16 decoder which can possibly be high due to the function of the AND gate.

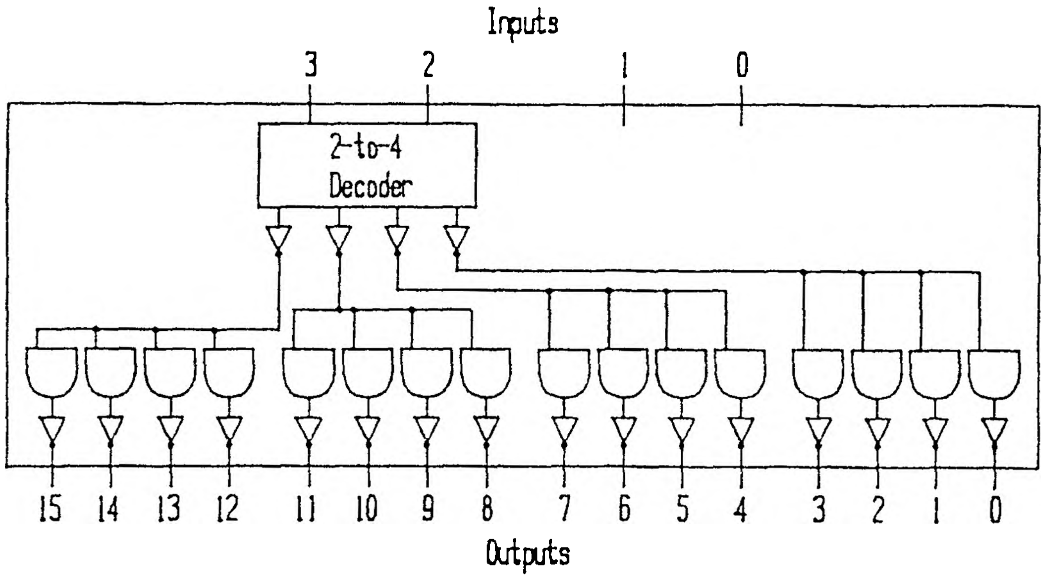


Figure 2.23. Partial 4-to-16 Decoder Using a 2-to-4 Decoder

Once one subset of four decoder output lines has been selected from the original sixteen, it remains to choose one from among the four. This will require four selection lines which will act as inputs to the AND gates of the output lines. Two of the original inputs have been used to this point, leaving two others; four selection lines are required. Thus a second 2-to-4 decoder may be used. Its inverted outputs will run in parallel to each of the subsets of four decoder outputs; see Figure 2.24. In this fashion a subset is chosen by the first 2-to-4 decoder, and a line from among the subset is selected by the second 2-to-4 decoder.

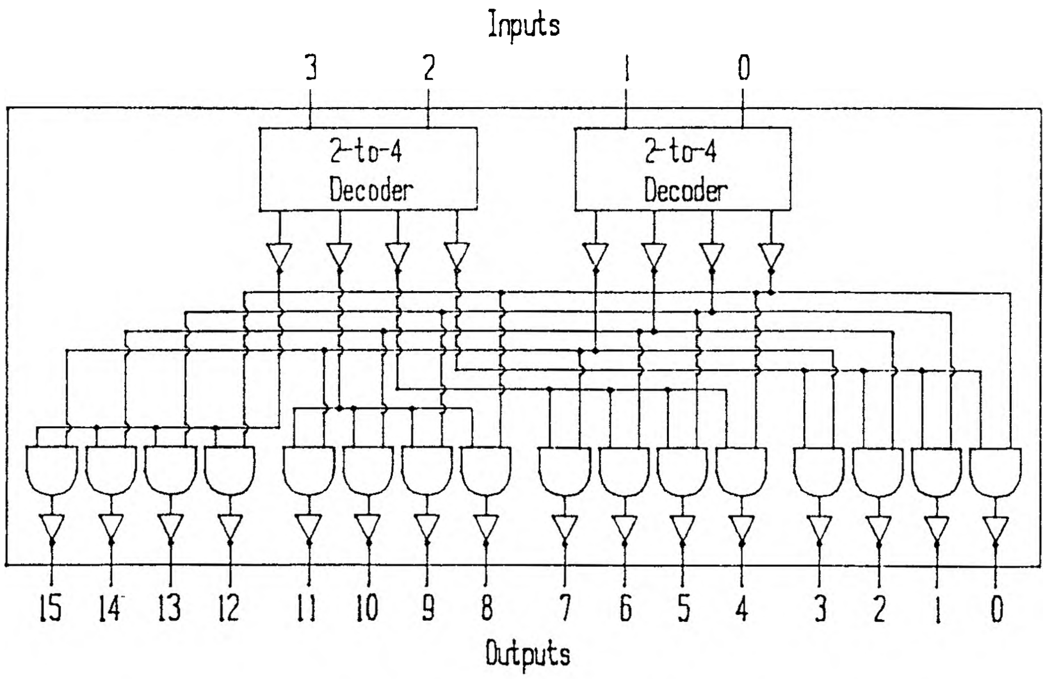


Figure 2.24. 4-to-16 Decoder Incorporating Two 2-to-4 Decoders

The second version of the 4-to-16 decoder can provide a reduction in the number of parts required by the first version. Assume that only transistor-based gates are used. In the first version of the decoder, twenty inverters were used along with sixteen 4-input AND gates, which were seen in Figure 2.21 to incorporate four transistors each. Thus a total of twenty inverters and sixty-four gate transistors were required. At one transistor per inverter, as seen in Figure 2.16, the transistor total becomes eighty-four. In the second version, there are two 2-to-4 decoders, twenty-four inverters, and sixteen 2-input AND gates. Figure 2.19 shows that each of these AND gates requires only two transistors, for a total of thirty-two. The 2-to-4 decoders, depicted in Figure 2.20, are each comprised of six inverters and four 2-input AND gates. Two such decoders will call for twelve inverters and eight 2-input AND gates, which also require two transistors each for a total of sixteen. Thus this version of the decoder requires thirty-six inverters and forty-eight gate transistors. Again at one transistor per inverter, the transistor total for the complete decoder is eighty-four, the same as before. However, this number is unnecccessarily large. Inspection of Figure 2.24 shows that the 2-to-4 decoder outputs are immediately inverted, while Figure 2.20 shows that the 2-to-4 decoder outputs are inverted immediately prior to being output. Thus the inverters on both sides of the two 2-to-4 decoders may be removed, as their actions nullify one another. There are four inverters on either side of both 2-to-4 decoders, for a total of sixteen. Consequently, sixteen transistors may be eliminated from the count for the second 4-to-16 decoder, bringing its total to sixty-eight. This is close to a 20% reduction from the first version. By reducing the number of components required the cost of the device is lessened, while its dependability is increased as there are fewer components to break down.

There are drawbacks to this second method, however. One trade off that is made here is with regard to the time it takes for the device to act. In Figure 2.22 the input lines of the first version of the decoder can be seen to pass through inverters and then proceed immediately to the 4-input AND gates. The second decoder construction has the inputs pass through the 2-to-4 decoders, the output of which continues on to the 2-input AND gates. Within the 2-to-4 decoders, the inputs pass through inverters and then another set of 2-input AND gates. Thus the second version is slower than the first version by the amount of time it takes a second AND gate to act. Regardless of whether the AND gate is diode-based or transistor-based, a finite amount of time is required to raise or lower the AND gate from the instant the inputs dictate a change. At computer speeds, this is an important aspect which cannot be ignored. On the other hand, with the project under development this amount of time is inconsequential; whether a robot joint begins moving a few thousandths of a second later or not will have no impact on the final outcome in this situation.

The actual device used for the transformation from the eight available output lines to the twelve required is the 74154 4-to-16 decoder. The 74154 decoder is a 24-pin integrated circuit which incorporates in a single package all of the necessary inverters and AND gates specified; see Figure 2.25 for the layout of the pins, or circuit connections. Pins 20, 21, 22, and 23 are the inputs described earlier as inputs 3, 2, 1, and 0; pin 20 is effectively the $2^3 = 8$'s place in the binary number being input, pin 21 is the 4's place, pin 22 is the 2's place, and pin 23 is the 1's place. With four binary positions, a value from 0 to 15 may be represented. High voltages (+5 volts) are interpreted as ones, while low voltages (0 volts) act as zeroes. Thus the decimal input 8 would appear as a high voltage at pin 20 and low voltages at pins 21, 22, and 23. Pins 18 and 19 act as enabling pins for the integrated circuit. In order for it to function as described, the voltages at both of these pins must be low. For this project, these two pins will be connected directly to ground.

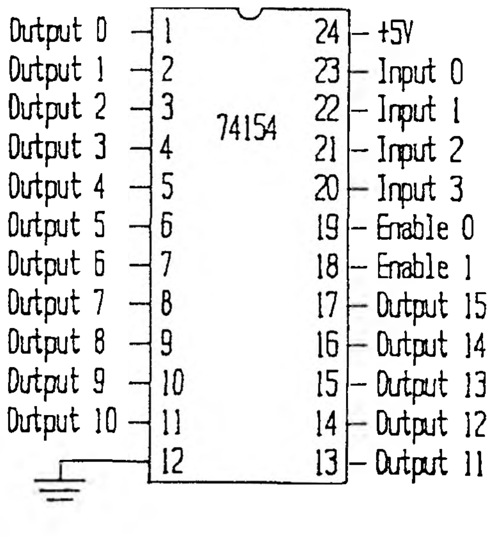


Figure 2.25. The 74154 4-to-16 Decoder

The 74154 IC maintains sixteen output lines, fifteen of which are at any one time at a high voltage level due to the grounding of pins 18 and 19; the permanent low voltages at pins 18 and 19 lead to the decoder being enabled at all times, with a sixteenth output, determined by the input value, low at all times. Pins 1 through 11 and 13 through 17 correspond to the outputs 0 through 15 in the development of the 4-to-16 decoder. It is arbitrarily decided here that the twelve transistor control lines will be obtained from the decoder's outputs of 1 through 12; the corresponding IC pins are 2 through 11, 13, and 14. When a joint is to be moving, the decoder inputs will indicate the output control line to the appropriate motor. At times when no joint is to be moving, some output must be selected besides those associated with transistor control. As only twelve of the sixteen outputs are needed for motor control, any of the remaining four may be set aside to be low when no motor is to be in use; let this be decoder output 0, corresponding to IC pin 1. If the situation called for the use of all sixteen decoder outputs for control lines, the enable lines could be employed to place high voltages on all sixteen lines simultaneously. As this is not the case, the approach taken is satisfactory. The remaining decoder output pins, 15, 16, and 17, are left unused. Note also that four of the eight available computer output lines are left unused. The remaining pins of the 74154 IC are pin 24, which receives a +5 volt power source and pin 12, which goes to ground. The 4.5 volt power supply of the circuit in Figure 2.5 is sufficient for operation of the IC; it will be supplied to pin 12.

There is an additional point that should be made about this arrangement. If the computer used had the necessary twelve control lines and they were tied directly to the resistors at the base of each transistor, then each of the transistors could be controlled independently of the others. This is an advantage in that it allows for the programmed control of coordinated motion, wherein more than one joint is moving at a time. This would make possible smoother motions by the robot arm. It is also a disadvantage in that if the control lines leading to two transistors connected to the same motor were made low simultaneously, then they would both allow current to flow; this current would proceed directly from the 9 volt source through both transistors to ground, as the emitter of the first transistor is tied to the collector of the second. This would very quickly ruin both transistors. One of two courses of action would need to be taken. First, this situation could be programmed against, a costly step that would have to be present in every application. The alternative would be to add additional circuitry which would preclude the two lines in question from going low during the same period. This has the disadvantage of adding some cost and complexity to the design. With the use of four control lines from the computer and the 74154 decoder's twelve subsequent outputs, it is impossible for more than one of the twelve lines to be low at a time. While coordinated motion is prevented with this design, the safety of the transistors is never an issue.

6. Inverters. While there are now twelve control lines for the twelve transistor bases, one final problem remains: the current provided by an output of the 74154 decoder is not sufficient to enable a 2N2907A pnp transistor to conduct current from emitter to conductor. Recall that when the base voltage is made low, the emitter-base junction becomes forward-biased, and current begins to flow. Holes leave the emitter, enter the base, and are then immediately attracted toward the collector's negative terminal. However, some of the holes entering the base from the emitter will recombine with the free electrons present in the n-type material. These electrons must be replaced by the base current. In the given situation the base current cannot be provided by the decoder as that source by itself is insufficient.

One method of handling this problem involves the use of the inverter described in the development of the decoder. The inverter depicted in Figure 2.16 was seen to incorporate a transistor and two resistors. The inverter circuit may be found separately on integrated circuits as the inverter is a common circuit element. In fact, six such inverters may be found on the 7404 IC, aptly referred to as a hex (six) inverter; see

Figure 2.26 for the layout of the integrated circuit. The first inverter can be seen to receive its input at pin 1 and produce its output at pin 2; subsequent inverter input and output pairs are pins 3 and 4, 5 and 6, 9 and 8, 11 and 10, and 13 and 12. Upon the presence of a high voltage at an input, the voltage at the corresponding output pin becomes low, and vice versa. The remaining pins of the 7404 IC are pin 14, which receives a +5 volt power source (detailed in Figure 2.5) and pin 7, which goes to ground. As was the case for the 74154 IC, the 4.5 volt power source of Figure 2.5 is sufficient for proper operation of the 7404 IC.

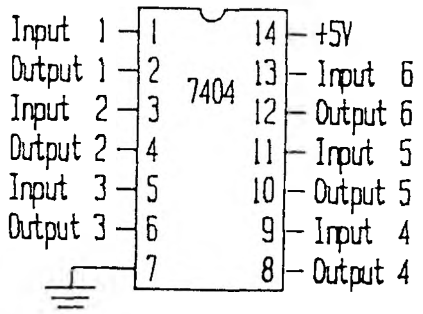


Figure 2.26. The 7404 Hex Inverter

Inverting the decoder output once certainly does not solve the problem as the voltage level produced is made opposite of that desired. Inverting the output of the first inverter removes this aspect of the problem by returning the voltage to its original level. However, the output of this second inverter still does not provide the required current. To obtain the amount needed, the output of the first inverter is passed through not one but three parallel inverters. The output currents of three 7404 inverters combined is adequate for the operation of the 2N2907A transistor. Each decoder will pass through one inverter followed by three inverters in parallel. With four inverters required for each of the twelve decoder outputs, a total of forty-eight inverters will be necessary. As a 7404 integrated circuit contains six inverters, a total of eight 7404 IC's will be used. Of course, the order of the connections among the inverter IC's is completely arbitrary. The design used in Figure 2.27 calls for the twelve decoder outputs to be connected to the inverters of a pair of 7404 IC's; the twelve subsequent inverter outputs are then passed two at a time to the remaining six IC's, each line going to three more inverters on a single IC. The three resultant parallel outputs may then be connected to the resistors at the base of each transistor.

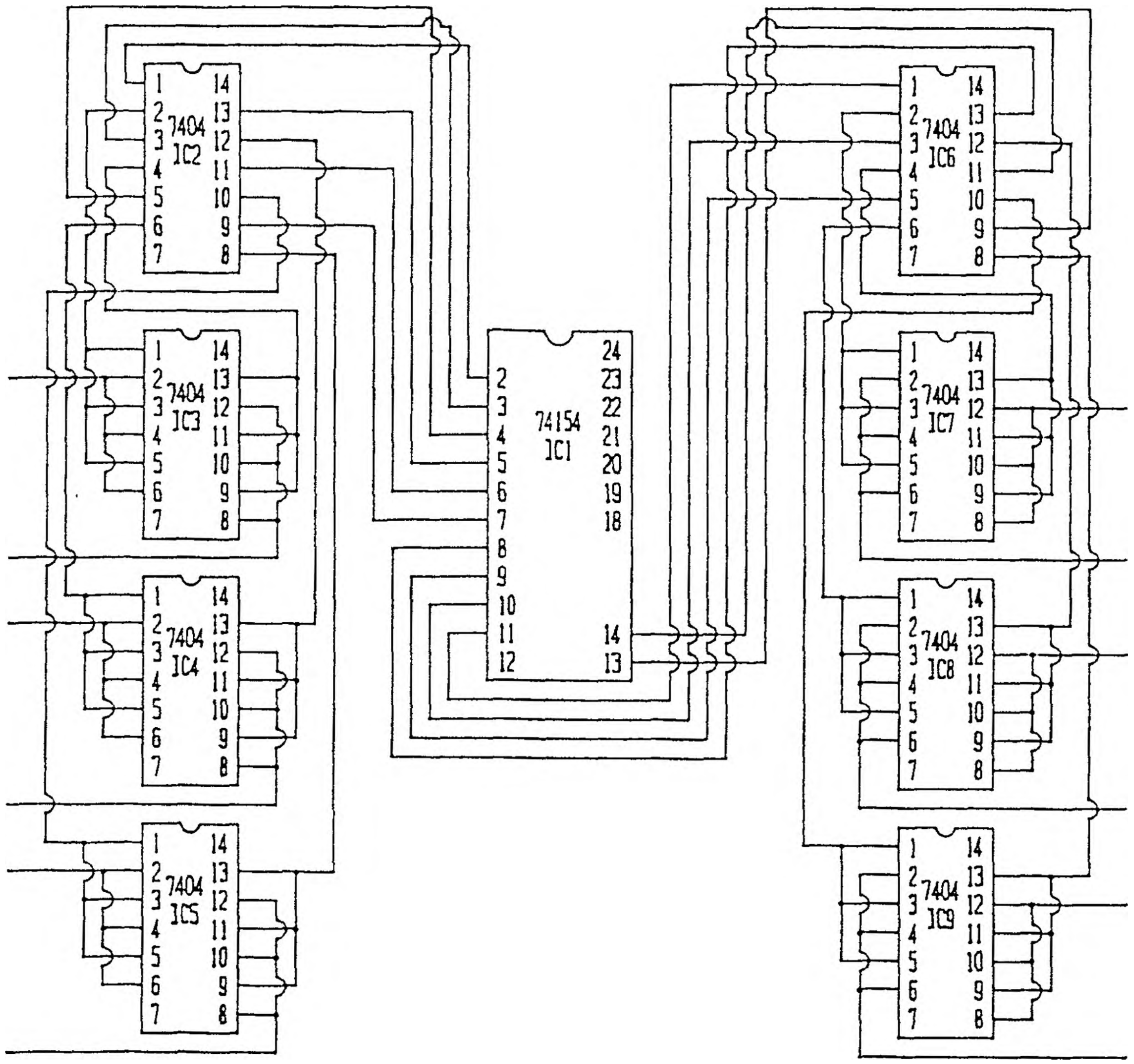


Figure 2.27. Decoder and Inverter IC Connections

At this point, the electronic interface of computer and robot motors is complete. The combined circuit can be seen in Figure 2.28. The subcircuits for the motor and integrated circuit power supplies were given earlier in Figure 2.4 and 2.5, respectively.

The actual construction was carried out on a 2200-hole 4.5" by 6.625" grid board. Sockets were used for all of the transistors and resistors, as well as the IC's, in order to facilitate replacement.

7. Manual Switches. One final segment was added to the electronic interface and remains to be described. The circuit given thus far provides for computer control only. In order for the robot to be moved manually, a set of switches is incorporated to provide the motors with power directly from the power supplies; see Figure 2.29. The twelve switches are of the momentary push button variety and are normally open. As can be seen in the figure, one side of each motor is fixed at the +4.5 volt level. Closing the corresponding top-row switch places the other side of the motor at ground, and current flows in one direction; closing the corresponding bottom-row switch places the other side of the motor at the +9 volt level, causing current to flow in the opposite direction. These switches are useful for alignment of the robot arm prior to programmed control. Care must be taken not to close both of the switches for a single motor simultaneously as this would put the +9 volt source directly in contact with the ground.

This concludes the construction of the robot arm for this project. The completed robot with electronic interface in place was shown earlier in Figure 2.1. Subsequent chapters will deal with various types of programmed control.

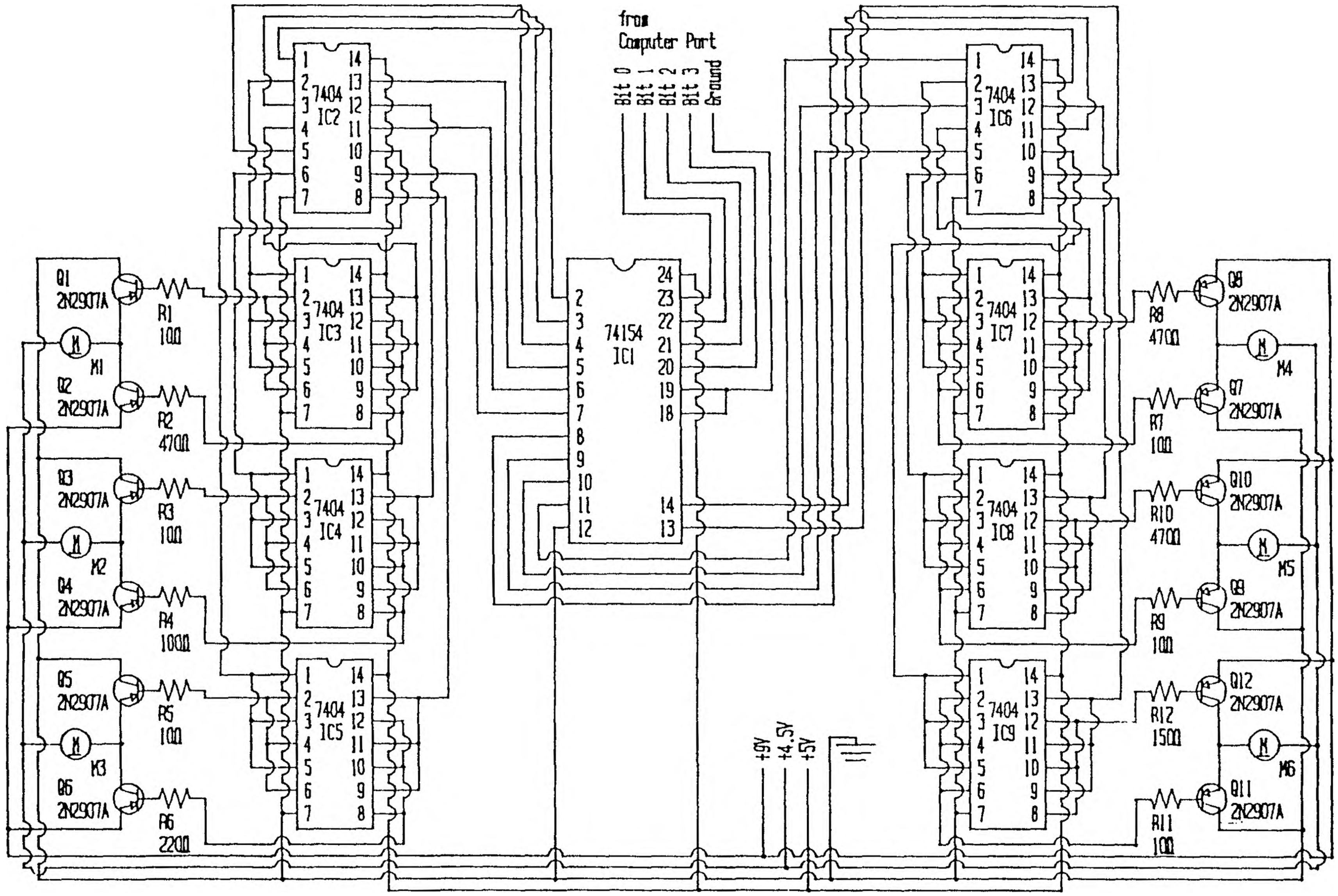


Figure 2.28. The Complete Computer Control Circuit

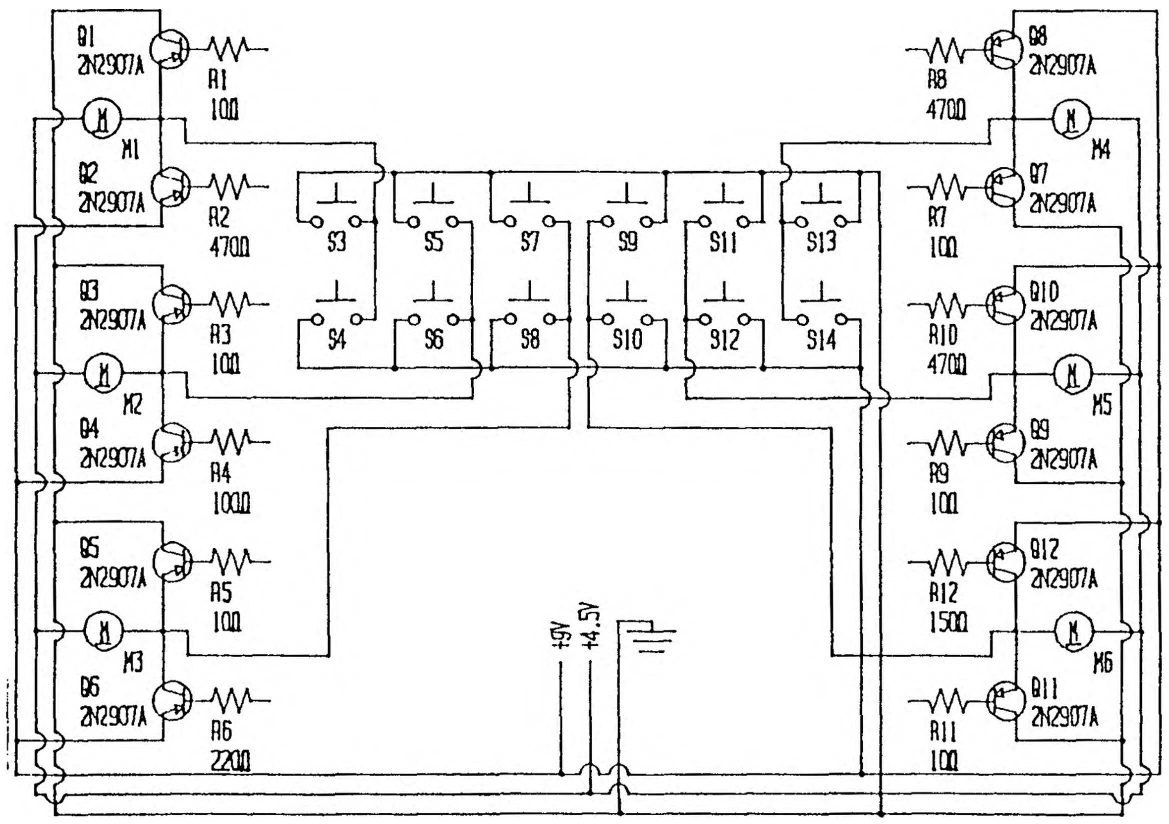


Figure 2.29. Switches for Manual Control

III. MANUAL CONTROL OF THE ROBOT

With the robot constructed as described in Chapter II, computer control can now be achieved. This chapter will detail the use of four control lines from the output port of the computer and how to actuate a selected joint via keyboard input using a simple control program.

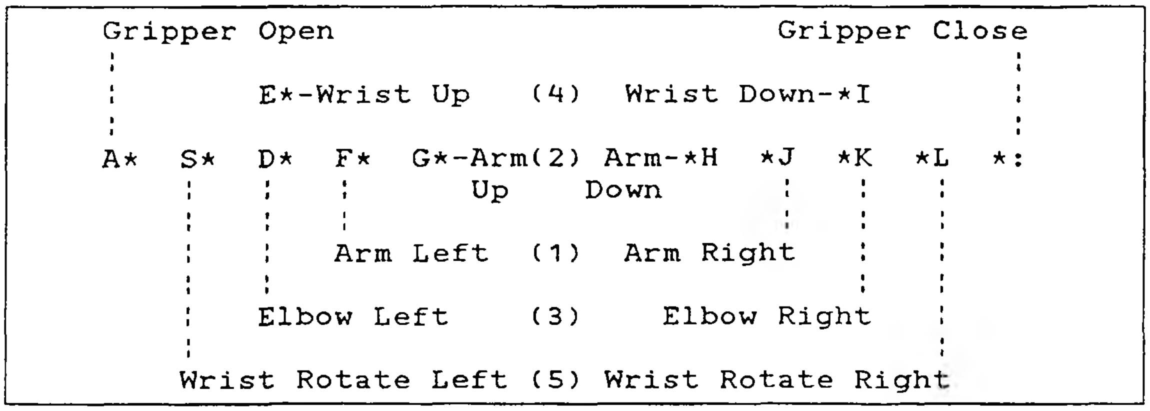
A. THE PARALLEL PORT

The IBM PC input/output port addressing scheme places parallel printer 2 at address 378H through 37BH. The eight data lines themselves are at location 378H, or 888. This is the position that will be addressed for setting the robot control lines. The other miscellaneous lines associated with printer control are not needed for this application. The C procedure OUTPORTB places a byte at the specified output port. The command `outportb (888, 0)` places all zeroes on the data lines. This command is issue at the beginning of the highest level program procedure, prior to supplying power to the robot. Values of 1 through 12 may subsequently be placed in the second parameter position to turn on the various transistors in the robot control circuit.

B. KEYBOARD INPUTS

The keyboard of the IBM PC will be used in this first control program. One key will be associated with each of the twelve distinct members of the six joint-direction pairs. Two choices present themselves for control. First, the joint could begin movement upon the press of a key and stop at the press of another. The alternative is to begin movement when a key is depressed and continue movement until it is released. This second technique was chosen as it was deemed easier to use and more effective in stopping control.

The key to joint-direction assignments are made arbitrarily. It was decided that a joint should be controlled in its two directions by keys of the same finger on each hand. For example, the left forefinger's "F" will drive the arm left while the right forefinger's "J" will cause a movement to the right. The entire set of assignments is presented to the user by a constant screen indicating them. The relevant portion of the display may be seen in Figure 3.1.



```
Gripper Open                          Gripper Close
:                                                 :
:         E*-Wrist Up   (4)  Wrist Down-*I        :
:                                                 :
A*   S*   D*   F*   G*-Arm(2) Arm-*H   *J   *K   *L   *:
     :    :    :       Up      Down     :    :    :
     :    :    :                        :    :    :
     :    :    Arm Left    (1)  Arm Right    :    :
     :    :                                  :    :
     :    Elbow Left     (3)     Elbow Right      :
     :                                            :
     Wrist Rotate Left (5) Wrist Rotate Right
```

Figure 3.1. Displayed Assignments of Keys to Joints

The procedure which polls the keyboard, **monitor_keyboard**, iterates repeatedly until the space bar is struck, the arbitrarily selected indication that manual control is to be terminated.

```
init_transistor_messages (msgs);
do
   {
   locate (row, col);
   while ( !kbhit( ) );
   key = toupper(getch( ));
   switch (key)
      {
                                    /* Arm Down - Up */
      case 'H' : transistor =  7;
                 joint = 2;
                 break;
      case 'G' : transistor =  8;
                 joint = 2;
                 break;
```

```
                                            /* Arm Right - Left */
        case 'J' : transistor = 12;
                   joint = 1;
                   break;
        case 'F' : transistor = 11;
                   joint = 1;
                   break;
                                            /* Elbow Left - Right */
        case 'D' : transistor =  6;
                   joint = 3;
                   break;
        case 'K' : transistor =  5;
                   joint = 3;
                   break;
                                            /* Wrist Down - Up */
        case 'I' : transistor =  1;
                   joint = 4;
                   break;
        case 'E' : transistor =  2;
                   joint = 4;
                   break;
                                            /* Wrist Rotate Left - Right */
        case 'S' : transistor =  9;
                   joint = 5;
                   break;
        case 'L' : transistor = 10;
                   joint = 5;
                   break;
                                            /* Gripper Open - Close */
        case 'A' : transistor =  4;
                   joint = 0;
                   break;
        case ';' : transistor =  3;
                   joint = 0;
                   break;

        default  : transistor =  0;
                   joint = 0;
        }
     if ( transistor != 0 )
        move_manual (transistor, msgs[transistor], key, row, col,
                      theta, joint);
     if ( joint > 0 )
        noap_matrix (theta, noap, noap_row, noap_cols);
     }
     while ( key != space );
```

Upon the pressing of a key associated with robot control, the procedure selects the appropriate joint and transistor; the procedure **move_manual** of the next section is then invoked to raise the appropriate control lines while the key is depressed. Procedure **noap_matrix**, which will be explained fully in the next chapter, is then invoked to update both the displayed joint values and the gripper position and orientation to reflect the completed movement. The position and orientation are updated for changes in joint variables but not gripper openings and closings.

C. ACTUATION OF A ROBOTIC MOTOR

When a key on the IBM PC keyboard is pressed, the character associated with the key is placed in a buffer for processing. The C procedure `kbhit` returns a true value when this buffer is not empty. There are two characteristics of the IBM PC keyboard which must be taken into account when implementing robotic control. First, when a key is pressed and held down, the keyboard microprocessor immediately responds by sending one instance of the associated character to the keyboard buffer; if the key remains depressed for longer than the period IBM refers to as the typematic delay, the keyboard microprocessor will begin to send additional copies of the character. The typematic delay for the IBM PC is 0.5 seconds; later keyboard models feature a programmable typematic delay, but the default remained 0.5 seconds. The second keyboard characteristic which must be dealt with is the repeat rate at which the keyboard processor sends out copies of the key character while it remains depressed. This is a programmable feature of later keyboard models. A rate of about 10 characters per second was standard for early keyboards and is the default value for later models. Information pertaining to the behavior of IBM equipment was obtained from Norton and Wilton [Nort88].

The specified control method calls for the selected drive motor to have power applied when the corresponding key is depressed; the power is to be discontinued when the key is released. Taking the typematic delay into account, the keyboard buffer cannot be examined again until 0.5 seconds have passed. Thus, there will be a lag of 0.5 seconds between the press of a key and the start of a motor. After that amount of time, the keyboard buffer may be examined about every 0.1 seconds to see if the key remains depressed. The loop structure of procedure `move_manual` takes these factors into account.

```
    degree_scale = select_scale_manual_move (transistor);
    i = 0;
```

```
lcputs (row, col, msg);
locate (row, col);
pause (500);
outportb (888, transistor);
do
   {
   i++;
   pause (110);
   key = null;
   while (kbhit ( ))
      key = toupper (getch ( ));
   }
   while ( key == original_key );
outportb (888, 0);
lcputs (row, col, "                    ");
if ( joint > 0 )
   theta[joint] += (float) i / degree_scale;
```

Experimentation with the given loop structure found that the execution time of the loop itself plus an additional 0.1 seconds was slightly less than the realized repeat rate. An increase in the loop's pause to 0.11 seconds was found to be satisfactory. Note that the keyboard buffer is emptied on each pass of the loop in the event that the key is released and several others are struck in rapid succession.

One other point about the procedure concerns the incrementation of variable `i` on each iteration of the loop. This is done to keep track of the current setting of each joint variable. Experimentation with the loop was performed to determine the approximate number of degrees the different joints turned in each direction during each pass of the loop. This process will be examined more fully in the next chapter, where the joint and axis assignments are discussed. The results of the experimentation are shown in Table 3.1. The number of iterations divided by the number of iterations per degree results in the number of degrees turned by the joint during the move. This value is then added to the previous setting of the joint to obtain its current setting.

D. THE CONTROLLING PROCEDURE

Procedure **manual_control** begins by displaying the introductory screen of Figure 3.2. Subsequent to that, the keyboard assignments in Figure 3.3 are displayed.

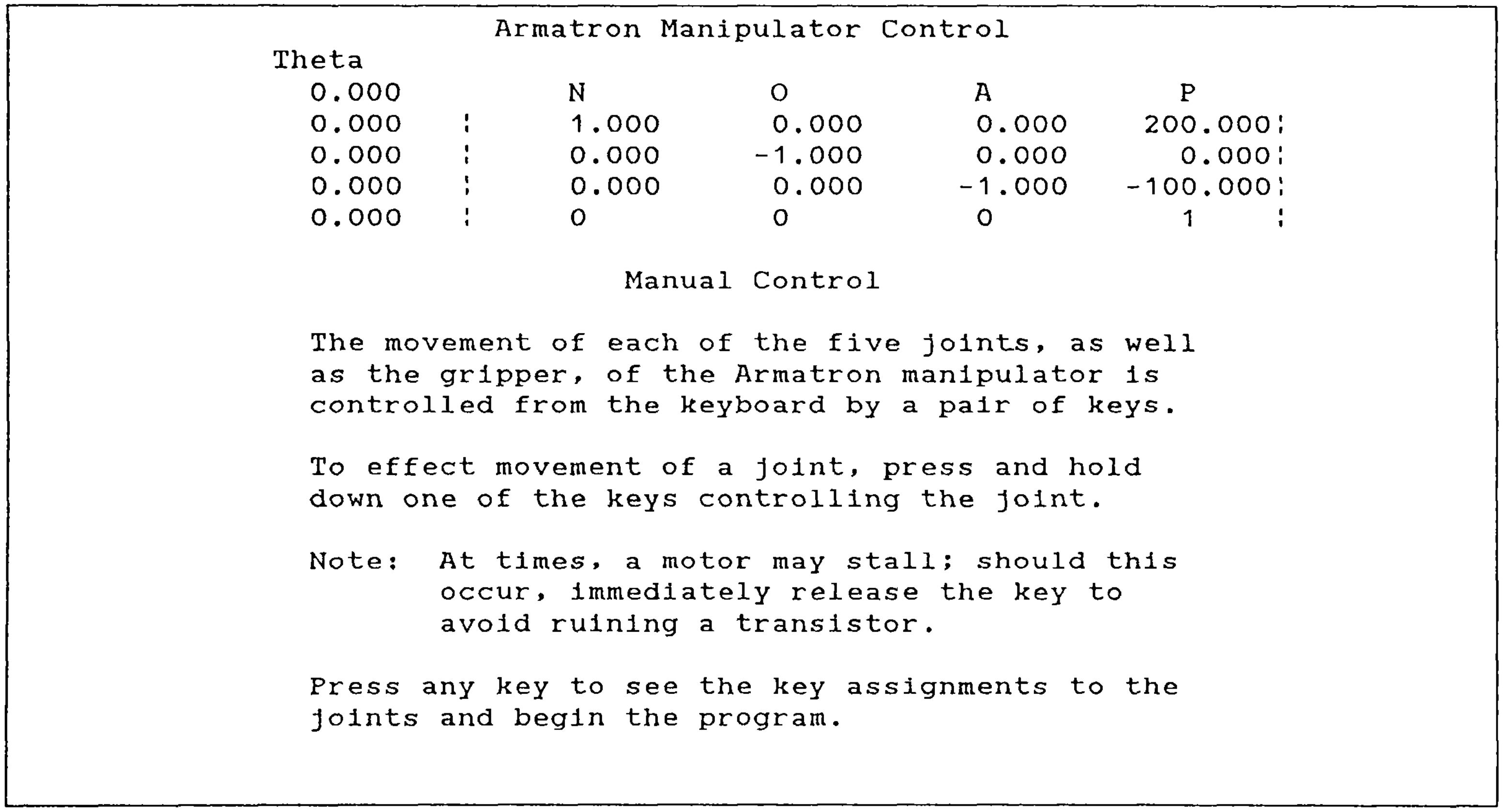
Table 3.1. RATIOS OF ITERATION COUNTS TO DEGREES MOVED

Joint	Direction	Ratio
Arm	Left	710 / 360
Arm	Right	650 / 360
Arm	Down	290 / 35
Arm	Up	270 / 35
Elbow	Right	415 / 180
Elbow	Left	410 / 180
Wrist	Up	365 / 200
Wrist	Down	300 / 200
Wrist-Rotate	Left	680 /1080
Wrist-Rotate	Right	660 /1080

The keyboard monitoring procedure is then invoked and retains control until manual control is to be terminated.

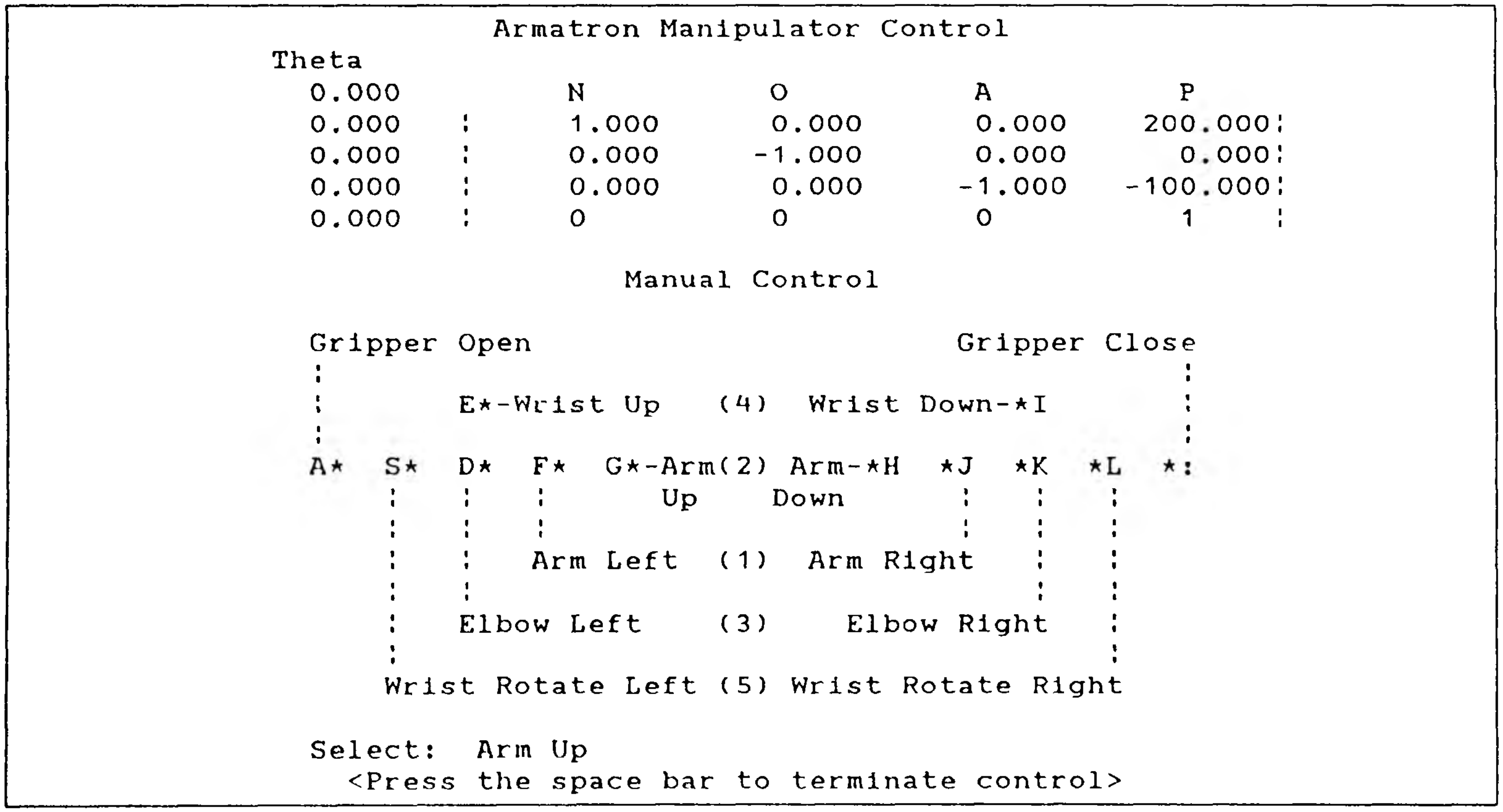
```
dsply_manual_introduction ( );
wait_then_erase (9);
dsply_keyboard (&row, &col);
monitor_keyboard (theta, noap, noap_row, noap_cols, row, col);
erase_prompt (23);
locate (23, 0);
mcputs (28, 27, "Manual Control Terminated");
wait_then_erase (8);
```

Note that Figure 3.3 depicts the display while the key "G" is depressed; a message is displayed on the select line reflecting this. The documented listing for the procedures associated with the manual control portion of the overall program many be found in Appendix C.



```
                  Armatron Manipulator Control
      Theta
        0.000             N          O          A          P
        0.000    |     1.000      0.000      0.000    200.000|
        0.000    |     0.000     -1.000      0.000      0.000|
        0.000    |     0.000      0.000     -1.000   -100.000|
        0.000    |     0             0          0          1    |

                         Manual Control

        The movement of each of the five joints, as well
        as the gripper, of the Armatron manipulator is
        controlled from the keyboard by a pair of keys.

        To effect movement of a joint, press and hold
        down one of the keys controlling the joint.

        Note:  At times, a motor may stall; should this
               occur, immediately release the key to
               avoid ruining a transistor.

        Press any key to see the key assignments to the
        joints and begin the program.
```

Figure 3.2. Manual Control Introduction



```
            Armatron Manipulator Control
Theta
  0.000         N          O          A          P
  0.000   :     1.000      0.000      0.000    200.000:
  0.000   :     0.000     -1.000      0.000      0.000:
  0.000   :     0.000      0.000     -1.000   -100.000:
  0.000   :     0          0          0          1    :

                   Manual Control

  Gripper Open                       Gripper Close
  :                                              :
  :       E*-Wrist Up   (4)  Wrist Down-*I       :
  :                                              :
  A*  S*  D*  F*  G*-Arm(2) Arm-*H  *J  *K  *L  *:
      :   :   :      Up    Down      :   :   :
      :   :   :                      :   :   :
      :   :   Arm Left  (1)  Arm Right   :   :
      :   :                              :   :
      :   Elbow Left    (3)    Elbow Right   :
      :                                      :
      Wrist Rotate Left (5) Wrist Rotate Right

  Select:  Arm Up
    <Press the space bar to terminate control>
```

Figure 3.3. Manual Control Display

IV. JOINT VARIABLE CONTROL

The previous chapter developed a program which would allow manual control of robot joints via a keyboard. While this demonstrated what is required of a computer to drive the robot motors, it did not result in a practical method of automated control. This chapter takes a step in this direction by developing a control program which provides for the specification of angular settings for each of the joints. The objective here will be to position and orient the arm prior to any actuation of the gripper, so control for the opening and closing of the gripper itself will be ignored. In addition to providing control, the position and orientation of the resulting arm configuration will be mathematically derived and displayed.

A. INITIAL ARM CONFIGURATION

In order to move a robot arm joint from one position to another, the control program will maintain a variable associated with the joint for the current setting of the joint angle. When a new setting is desired, comparison with the joint variable will yield the amount of movement required. The arm will have an initial configuration at which each of the joint variables will be zero. This initial position and orientation will dictate the maximum positive and negative movements possible for each of the joints. Choosing an initial configuration is done with the primary goal of simplifying subsequent calculations involving the arm's position and orientation.

Consider the depiction of the Armatron manipulator configuration shown in Figure 4.1. This is the initial configuration to be used. Note that each of the five joints has an axis indicated about which it can rotate. The base coordinate frame for specifying the position and orientation to the user by vectors is indicated by the vector-triple ($\vec{x}_0$, $\vec{y}_0$, $\vec{z}_0$). The origin of this coordinate-frame is located at the intersection of the axes of rotation of the first two joints, the arm-vertical and the arm-horizontal rotations. There are two things to note here. First, the direction for the axis of rotation for each joint has two alternatives; for example, the axis about which the vertical arm rotation takes place could extend to the right, rather than the left as in Figure 4.1. Second, the configuration has the arm fully extended horizontally, but the wrist is pointed vertically down. The reason for the axis directions used and the initial configuration will be made clear later in this chapter.

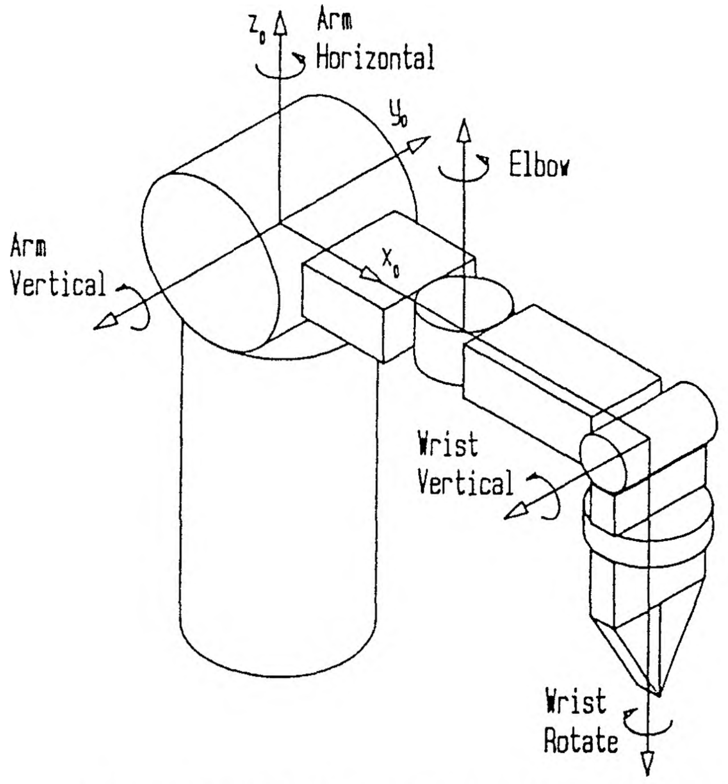


Figure 4.1. Initial Configuration of the Robot Arm

Each of the five joints has restraints placed on it by the physical construction of the robot. For example, the wrist is capable of upward and downward movement throughout a 200° arc. The initial configuration divides each arc of movement into positive and negative portions. The wrist reaches the center of its arc of vertical rotation when it is extended horizontally. As the initial configuration calls for the wrist to be pointed straight down, only 10 more degrees remain in the downward direction until the bottom of the arc of movement is reached; the 190° complement is available in the upward direction. One can apply the right-hand rule to the rotation axes of Figure 4.1 to define which movements are positive and which are negative. By placing one's right hand about the axis of rotation with the thumb extended along the positive direction of the axis, the fingers curl about the axis in the positive direction of movement. Thus, of the wrist's 200 degrees of movement, 190 are in the positive direction and 10 in the negative. The five joints, their arcs of movement, and the positive and negative movements are as stated in Table 4.I.

Table 4.I. RANGE AND DIVISION OF JOINT ARC MOVEMENTS

Joint	Arc	Positive	Negative
Arm-Horizontal	continuous	360 left	360 right
Arm-Vertical	35	30 up	5 down
Elbow-Horizontal	180	90 left	90 right
Wrist-Vertical	200	190 up	10 down
Wrist-Rotate	continuous	360 right	360 left

Note that the designations of left and right are with respect to the base-coordinate frame, looking out along the *x*-axis. Note also that the two joints having unlimited movement are arbitrarily restricted to 360° in either direction, as one full rotation is sufficient to obtain any position.

B. PROCESSING OF MOVE REQUESTS

A movement command for the manipulator consists of two parts, the joint to be driven and the associated number of degrees. With these two pieces of information known, the move may then be carried out. Procedure `process_requests` carries out these three steps repeatedly.

```
joint = get_joint ( );
while ( joint != 0 )
   {
   angle = get_angle (theta_min[joint], theta_max[joint]);
   erase_prompt (23);
   locate (23, 20);
cprintf ("Moving Joint %d to angle %8.3f", joint, angle);
perform_move (joint, theta, angle, jv_rows[joint], jv_col);
lcputs (23, 20, "                              ");

noap_matrix (theta, noap, noap_row, noap_cols);

joint = get_joint ( );
}
```

After each move, the joint variable values and the position and orientation of the manipulator's gripper will be updated by procedure **`noap_matrix.`** The next section will develop the relationship between the joint variables and the gripper position and orientation. This section examines each of the three steps directly involved with the move: selection of a joint, specification of an angle, and performance of the move.

1. Selection of a Joint. The first step in the process of joint variable control is to provide for the selection of the joint to be moved. The Armatron robot has five joints for movement of the arm. The joint numbering is indicated to the user along with the physical range for each in a display screen; the relevant portion is given in Figure 4.2.

```
          Joint
    1:  Arm Right/Left
        (-360 to +360)
    2:  Arm Down/Up
        (   -5 to  +30)
    3:  Elbow Right/Left
        ( -90 to  +90)
    4:  Wrist Down/Up
        ( -10 to +190)
    5:  Wrist Rotate Left/Right
        (-360 to +360)
```

Figure 4.2. Displayed Joint Numbering

The first program interaction is to obtain the number corresponding to the desired joint; procedure **`get_joint`** accomplishes this. The user is to choose a number between one and five. By convention, if an input of "0" is received, joint variable control is to terminate.

```
do
   joint = prompt_input_digit ("Select Joint:");
   while ( joint > 5 );
locate (24, 20);
cprintf ("Joint %d has been selected", joint);
return (joint);
```

The only way control is returned from the procedure to the invocation point is by obtaining a numeric digit of 0 through 5.

2. Specification of an Angle. With the joint chosen, a setting is required next. Procedure `get_angle` prompts for an input for the selected joint.

```
do
   {
   lcputs (23, 20, "Enter angle <Snnn.nnn>:  ");
   angle = (indec (23, 45));
   if ( angle == 1000 )
      angle = 0;
   if ( (angle < minimum) | (angle > maximum) )
      {
      locate (23, 10);
      cprintf ("Angle %8.3f out of range for the joint; ", angle);
      cputs ("check ranges above");
      pause (3000);
      locate (23, 10);
      cprintf ("%61c", ' ');
      }
   }
   while ( (angle < minimum) | (angle > maximum) );
return (angle);
```

Procedure `indec` does not allow the input of the integer portion of a value to exceed 999; the value 1000 is returned by it to indicate the entry of a null string, to which a value of 0 is assigned. Procedure `get_angle` then checks the input angle against the extreme values for the selected joint, iterating until a valid value is obtained.

3. Performance of the Move. Procedure `perform_move` carries out a move by determining a signed angle to move through, selecting the proper transistor, and then iterating an appropriate number of times. Each of these steps shall be examined in detail. The body of the procedure is presented here in its entirety; portions of it will

be pulled out and inspected during the examination of the steps for performing the move.

```
move_degrees = desired_position - theta[joint];
transistor = select_transistor (joint, move_degrees);
degree_scale = select_scale (transistor);
iterations = round (fabs(move_degrees) * degree_scale);
degrees_per_iteration = sign (move_degrees) / degree_scale;
lcputs (row, col, "    Moving");
i = 0;
outportb (888, transistor);
while ( (!kbhit( )) & (i < iterations) )
   i = i + 1;
outportb (888, 0);
if ( i == iterations )
      theta[joint] = desired_position;
   else
      {
      getch ( );
      theta[joint] += degrees_per_iteration * i;
      }
lcprintf (row, col, theta[joint]);
```

a. Determination of the Signed Angle. The array `theta` is used to maintain the current settings for each of the five joints. It is zeroed at the beginning of the main program, at which time the robot is aligned to its home positioning. The desired setting for the joint in question is input as `desired_position;` the required move is then the signed difference between `desired_position` and the value stored for the joint in the `theta` array. For example, if the desired setting is 45° and the current setting is 30°, a move of +15° is required; to attain 45° from a starting position of 60° requires a move of −15°. In both cases, the current setting is subtracted from that desired.

```
move_degrees = desired_position - theta[joint];
```

b. Selection of the Proper Transistor. Returning to the electronic interface configuration for a moment, it will be recalled that there are two control line-transistor pairs for each joint, one for each direction; thus if the specified angle of movement is positive, the forward control line shall be selected and the other if negative. This task is carried out by procedure `select_transistor`, which returns the transistor number based on the joint to be moved and the sign of the angle.

```
if ( move > 0 )
     switch (joint)
        {
        case 1 : transistor = 11;
                 break;
        case 2 : transistor =  8;
                 break;
        case 3 : transistor =  6;
                 break;
        case 4 : transistor =  2;
                 break;
        case 5 : transistor = 10;
        }
  else
     switch (joint)
        {
        case 1 : transistor = 12;
                 break;
        case 2 : transistor =  7;
                 break;
        case 3 : transistor =  5;
                 break;
        case 4 : transistor =  1;
                 break;
        case 5 : transistor =  9;
        }
return (transistor);
```

In the event that a move of zero is requested, the negative transistor for the joint is arbitrarily selected as there will not be any activity.

c. Iteration During the Move. There are two possibilities for controlling the movement of an arm joint through θ degrees. First, feedback hardware can be utilized to provide information to the computer as to the positioning of the arm; this method was deemed inappropriately complex with respect to the goals of this project. The other alternative is to control the joint movement by a timing scheme. The assumption here is that movement of a joint occurs at a constant fixed rate. Inertia, acceleration, and other complicating factors are outside the scope of this project and are thus ignored. Given this, a simple iterative loop can be used to control movement by first determining how many degrees of movement are obtained for each pass of the loop. The following loop structure was set up to make these determinations:

```
i = 0;
outportb (888, transistor);
while ( (!kbhit( )) & (i < iterations) )
```

```
     i = i + 1;
   outportb (888, 0);
```

The variable `iterations` was set to 10000, a value in excess of any iteration count to ever be used. The loop was then executed numerous times for each of the joints. The test for a joint begins by moving the joint manually to one end of its arc of movement. For example, the wrist is dropped to the lowest position possible. The variable `transistor` is set to the number of the transistor associated with the movement being tested. In the example, transistor 1 is used as it will cause upward movement of the wrist toward the highest position possible. The program segment is then run. The first statement raises the appropriate control lines, beginning movement. The loop itself first checks the keyboard buffer to see if any key has been struck; if not, control continues through the loop to the next pass. If a key has been struck, indicating that the wrist has achieved the maximum upward position, the loop is exited. The final statement lowers all control lines, ceasing movement. The value of `i` reflects the number of iterations required for the joint to move through its entire range in the given direction. In the case of the wrist, this range is 200 degrees, while the average iteration count was found to be 365. The inference from this is that the loop iterates 365/200 = 1.825 times for each degree of movement. This value of 1.825 is then the scale factor for the upward movement of the wrist joint. This scale factor can be used to determine the number of iterations necessary for any desired range of movement. For example, a movement of 100 degrees would require 100 degrees × 1.825 iterations/degree, or approximately 183 iterations of the given loop.

An additional point should be made about the loop structure. While the keyboard buffer test is obviously necessary for the scale-determination phase, it may seem that it should be removed from the loop before it is put into use; this would reduce the amount of time used on each iteration and in turn alter the count times as obtained. This point however is nullified as there are a pair of valid reasons for leaving this test in place. First, experimentation with the arm showed that a motor would occasionally stall; this led to the rapid burn-out of the selected transistor as current was not flowing through the motor. By retaining the ability to leave the loop early, the user can act as quickly as such a situation is noticed to save the transistor. Another advantage of this set up is that the user can abort a movement which is leading the arm to inadvertently strike another object in its envelope.

The geometric configuration of the arm gives the ranges in degrees for the joints as stated in Table 4.II.

Table 4.II. RANGES OF THE ARM JOINTS

Joint	Movement	Range
Arm	Horizontal	360
Arm	Vertical	35
Elbow	Horizontal	180
Wrist	Vertical	200
Wrist	Rotate	360

The tests carried out for each of the joints and directions produced the average iteration-count to degrees-moved ratios in Table 4.III. As wrist rotation can be performed continuously in the same direction, the counts were made for three complete rotations to improve accuracy; this was necessary due to the speed of the rotation. The arm movement from left to right is also continuous, but it is slow enough for one rotation to be sufficient for timing purposes.

The corresponding iteration-to-degree scales are returned by procedure `select_scale` which acts on the transistor number.

```
switch (transistor)
   {                                       /* Wrist Down - Up */
   case  1 : scale = 3000 /  200;
             break;
   case  2 : scale = 3650 /  200;
             break;
                                           /* Gripper Close - Open */
   case  3 : scale = 1;
             break;
   case  4 : scale = 1;
```

Table 4.III. ITERATION COUNTS TO DEGREES MOVED FOR JOINT CONTROL

Joint	Direction	Ratio
Arm	Left	710 / 360
Arm	Right	650 / 360
Arm	Down	290 / 35
Arm	Up	270 / 35
Elbow	Right	415 / 180
Elbow	Left	410 / 180
Wrist	Up	365 / 200
Wrist	Down	300 / 200
Wrist-Rotate	Left	680 /1080
Wrist-Rotate	Right	660 /1080

```
                break;
                                         /* Elbow Right - Left */
    case  5 : scale = 4150 /  180;
              break;
    case  6 : scale = 4100 /  180;
              break;
                                         /* Arm Down - Up */
    case  7 : scale = 2900 /   35;
              break;
    case  8 : scale = 2700 /   35;
              break;
                                         /* Wrist Rotate Left - Right */
    case  9 : scale = 6800 / 1080;
              break;
    case 10 : scale = 6600 / 1080;
              break;
                                         /* Arm Left - Right */
    case 11 : scale = 7100 /  360;
              break;
    case 12 : scale = 6500 /  360;
              break;
    }
  return (scale);
```

Note that transistors 3 and 4, which respectively control closing and opening of the gripper, have arbitrary scales of 1, as they will not be used.

The number of iterations for a given transistor and move are thus found by multiplication of the scale (iterations per degree) and the number of degrees to turn; additionally, the change in the joint per iteration is the inverse of the scale, or degrees per iteration, in the direction of the move to be made.

```
  degree_scale = select_scale (transistor);
  iterations = round (fabs(move_degrees) * degree_scale);
  degrees_per_iteration = sign (move_degrees) / degree_scale;
```

After the completion of the timing loop, the number of degrees moved is then the delta rate multiplied by the number of iterations; if the move was completed without interruption, a direct assignment is made to eliminate the error introduced by the scale division and subsequent multiplication.

```
if ( i == iterations )
      theta[joint] = desired_position;
   else
      {
      getch ( );
      theta[joint] += degrees_per_iteration * i;
      }
```

C. DETERMINATION OF POSITION AND ORIENTATION

The remainder of this chapter is dedicated to determining the position and orientation of the robot manipulator's gripper as the control variables change. The center point of the gripper closure will be derived as the components of the vector from the base coordinate frame to the center point. Associated with the gripper will be a triple of three unit vectors which will uniquely specify the orientation of the gripper; their vector components will also be obtained.

1. The Problem Approach. The problem of determining position and orientation in robotics is approached by considering the robot manipulator as a set of transformations. First, a base coordinate frame is established for the manipulator. The vector triple ($\bar{x}_0$, $\bar{y}_0$, $\bar{z}_0$) will represent this frame; see again Figure 4.1. World coordinates are specified with respect to the origin and directions of this frame. The manipulator will then have a series of relative coordinate frames established with respect to each of its joints. When a joint moves, it is transforming the next relative base frame and all those succeeding with respect to its own frame. The transformation due to a specific joint can be represented by a transformation matrix. The transformation matrices for all of the joints of the robot when multiplied together yield

a combined transformation matrix for the entire robot manipulator. Base-frame coordinates when multiplied by this single matrix will be transformed to the base-frame coordinates resulting from the series of rotations and translations. In this manner, a set of unit vectors in the x-, y-, and z-directions will be transformed to the x-, y-, and z-directions of the coordinate frame at the end of the manipulator after the movements; this will in turn specify how the gripper is oriented as the coordinate frame is fixed relative to the gripper. This transformation will also be done for the base coordinate frame origin to determine the center point of the gripper closure, as this is the coordinate frame origin at the manipulator's end.

2. Link Transformations. A link is said to connect one joint's coordinate frame to the next. By convention, link i covers the transformation from coordinate $i-1$ to i. The composition transformation accomplished over a single link with respect to the starting joint's coordinate frame of reference is by convention a sequence of four individual transformations. The transformations include a rotation angle θ_i about axis z_{i-1}, translation of a distance d_i along z_{i-1}, translation along axis x_{i-1} of distance a_i, and a rotation angle of α_i about x_{i-1} from the z_{i-1} vector direction to that of z_i. These four transformations allow for both the movement of the robot joints and the fixed offsets due to the physical dimensions of the robot. The link measures, or parameters, θ_i, d_i, a_i, and α_i are commonly presented in a link parameter table; such a table will be constructed for the Armatron manipulator.

Consider first the rotation θ_i about axis z_{i-1}. From Figure 4.3,

$$x = r\cos\phi \tag{4.1}$$

$$y = r\sin\phi \tag{4.2}$$

$$x' = r\cos(\phi + \theta_i) \tag{4.3}$$

$$y' = r \sin(\phi + \theta_i) \tag{4.4}$$

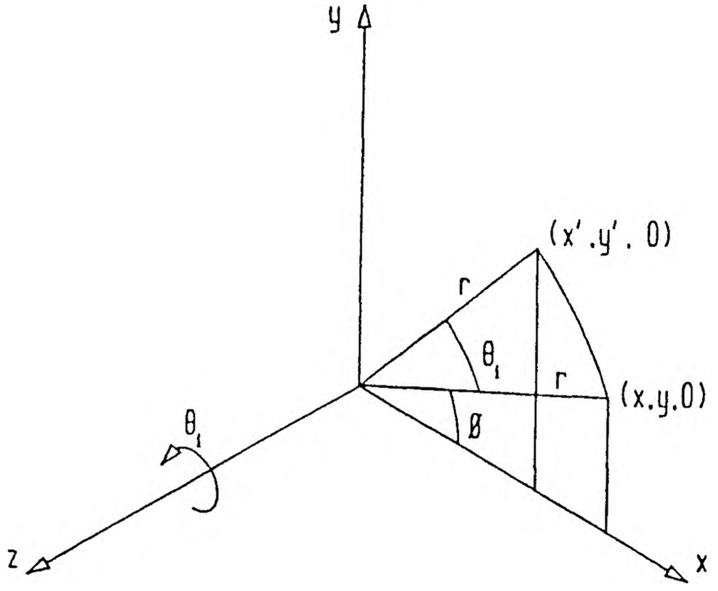


Figure 4.3. Rotation of a Point about the z-axis

Equations (4.3) and (4.4) are expanded using the trigonometric identities for the cosine and sine of the addition of two angles, respectively.

$$x' = r(\cos \phi \cos \theta_i - \sin \phi \sin \theta_i) \tag{4.5}$$

$$y' = r(\cos \phi \sin \theta_i + \sin \phi \cos \theta_i) \tag{4.6}$$

The right hand sides of Equations (4.5) and (4.6) are then regrouped to allow substitutions to be made from Equations (4.1) and (4.2).

$$x' = (r \cos \phi) \cos \theta_i - (r \sin \phi) \sin \theta_i \tag{4.7}$$

$$y' = (r \cos \phi) \sin \theta_i + (r \sin \phi) \cos \theta_i \tag{4.8}$$

$$x' = x \cos \theta_i - y \sin \theta_i \tag{4.9}$$

$$y' = x \sin \theta_i + y \cos \theta_i \tag{4.10}$$

Additionally, since the rotation was about a z-axis, there is no change in the z-coordinate.

$$z' = z \tag{4.11}$$

Equations (4.9), (4.10), and (4.11) may be combined into a single matrix equation.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.12}$$

The translation d_i along axis z_{i-1} of a point (x, y, z) results in the new point whose coordinates are given by the following equations:

$$x' = x \tag{4.13}$$

$$y' = y \tag{4.14}$$

$$z' = z + d_i \tag{4.15}$$

These equations may also be combined in matrix form.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.16}$$

Similarly, the translation a_i along axis x_{i-1} may be represented by the following equations.

$$x' = x + a_i \tag{4.17}$$

$$y' = y \tag{4.18}$$

$$z' = z \tag{4.19}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.20}$$

The final rotation α_i about the x_{i-1} axis is similar to that of the z-rotation and may be seen in Figure 4.4. From the figure,

$$y = r \cos \beta \tag{4.21}$$

$$z = r \sin \beta \tag{4.22}$$

$$y' = r \cos(\beta + \alpha_i) \tag{4.23}$$

$$z' = r \sin(\beta + \alpha_i) \tag{4.24}$$

Equations (4.23) and (4.24) are expanded as were (4.3) and (4.4) using the trigonometric identities for the cosine and sine of the addition of two angles, respectively, prior to regrouping and substitution from Equations (4.21) and (4.22).

$$y' = r(\cos \beta \cos \alpha_i - \sin \beta \sin \alpha_i) \tag{4.25}$$

$$z' = r(\cos \beta \sin \alpha_i + \sin \beta \cos \alpha_i) \tag{4.26}$$

$$y' = (r \cos \beta) \cos \alpha_i - (r \sin \beta) \sin \alpha_i \tag{4.27}$$

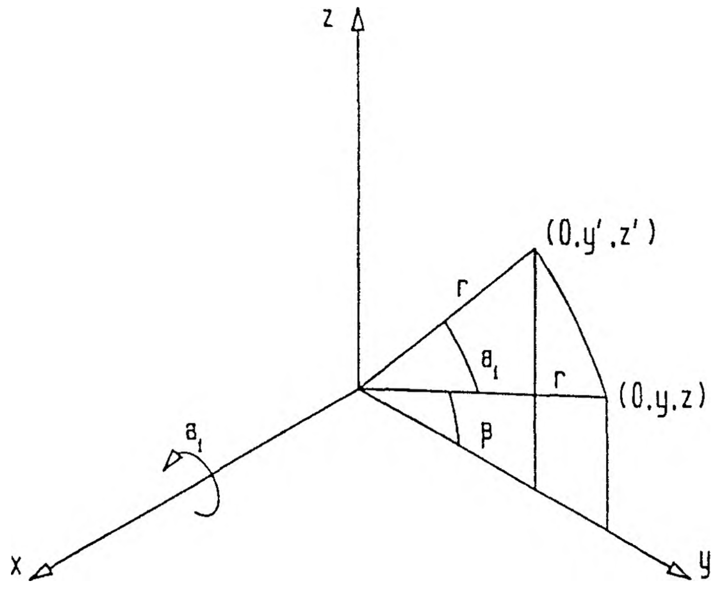


Figure 4.4. Rotation of a Point about the x-axis

$$z' = (r \cos \beta) \sin \alpha_i + (r \sin \beta) \cos \alpha_i \tag{4.28}$$

$$y' = y \cos \alpha_i - z \sin \alpha_i \tag{4.29}$$

$$z' = y \sin \alpha_i + z \cos \alpha_i \tag{4.30}$$

Since the rotation was about an x-axis, there is no change in the x-coordinate.

$$x' = x \tag{4.31}$$

Equations (4.31), (4.29), and (4.30) may also be combined into a single matrix equation.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha_i & -\sin \alpha_i & 0 \\ 0 & \sin \alpha_i & \cos \alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.32}$$

The effects of the four transformations in Equations (4.12), (4.16), (4.20), and (4.32) are then combined in a single equation.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.33}$$

The sequence of transformations performed on the point (x, y, z) by the matrix product of Equation (4.33) can be thought of in two ways. One can start with the original point and perform the x-rotation with respect to the base coordinate frame. This would be followed by an x-translation, again with respect to the base coordinate frame. The z-translation and z-rotation would successively follow, each still with respect to the base coordinate frame. One can visualize this using Equation (4.33) by successively multiplying the last two matrices, a transform and a point vector, step-by-step creating intermediate point vectors.

Consider a numeric example. Let the sequence of transformations be a rotation about the z-axis of 25°, a translation along the z-axis of 7 units, a translation along the x-axis of 16 units, and a rotation about the x-axis of 96°. Let the point to be transformed be $[22,\ 10,\ 13,\ 1]^T$. The process is performed in the reverse of this order, one transformation at a time. Figures 4.5 through 4.8 detail each of the steps graphically.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 96^\circ & -\sin 96^\circ & 0 \\ 0 & \sin 96^\circ & \cos 96^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 22 \\ 10 \\ 13 \\ 1 \end{bmatrix} \tag{4.34}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}' = \begin{bmatrix} 22.000 \\ -13.974 \\ 8.586 \\ 1 \end{bmatrix} \tag{4.35}$$

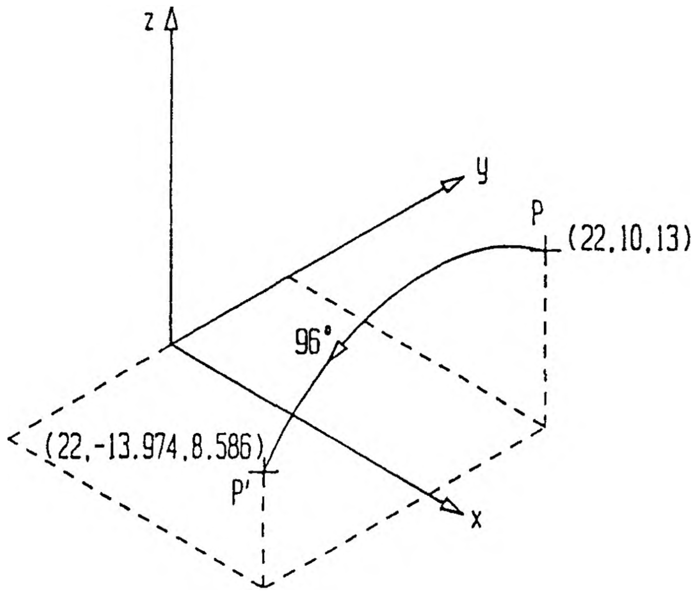


Figure 4.5. x-Rotation with Respect to the Base Coordinate Frame

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}'' = \begin{bmatrix} 1 & 0 & 0 & 16 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 22.000 \\ -13.974 \\ 8.586 \\ 1 \end{bmatrix} \tag{4.36}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}'' = \begin{bmatrix} 38.000 \\ -13.974 \\ 8.586 \\ 1 \end{bmatrix} \tag{4.37}$$

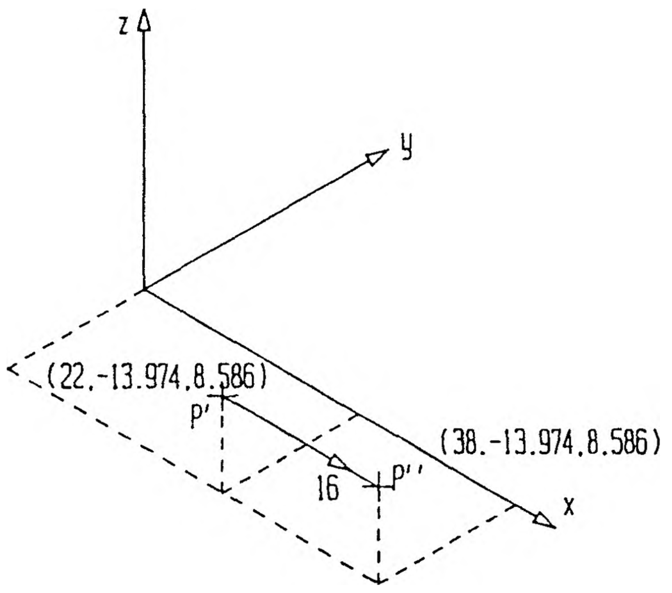


Figure 4.6. x-Translation with Respect to the Base Coordinate Frame

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{\prime\prime\prime} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 38.000 \\ -13.974 \\ 8.586 \\ 1 \end{bmatrix} \tag{4.38}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{\prime\prime\prime} = \begin{bmatrix} 38.000 \\ -13.974 \\ 15.586 \\ 1 \end{bmatrix} \tag{4.39}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{\prime\prime\prime\prime} = \begin{bmatrix} \cos 25^\circ & -\sin 25^\circ & 0 & 0 \\ \sin 25^\circ & \cos 25^\circ & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 38.000 \\ -13.974 \\ 15.586 \\ 1 \end{bmatrix} \tag{4.40}$$

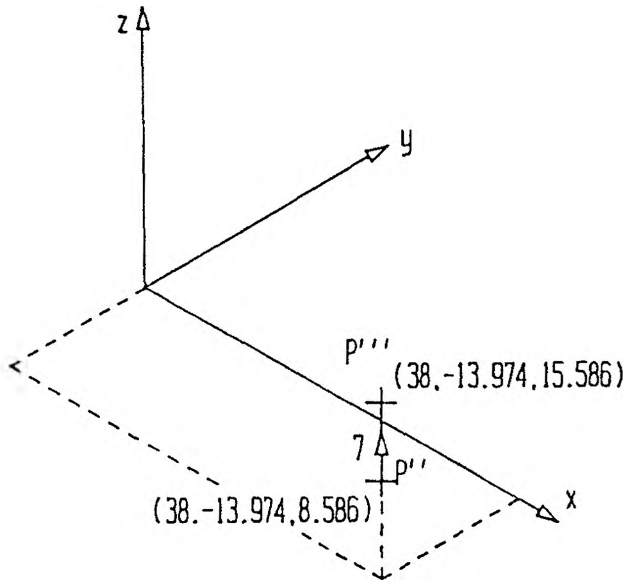


Figure 4.7. z-Translation with Respect to the Base Coordinate Frame

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{''''} = \begin{bmatrix} 40.345 \\ 3.395 \\ 15.586 \\ 1 \end{bmatrix} \tag{4.41}$$

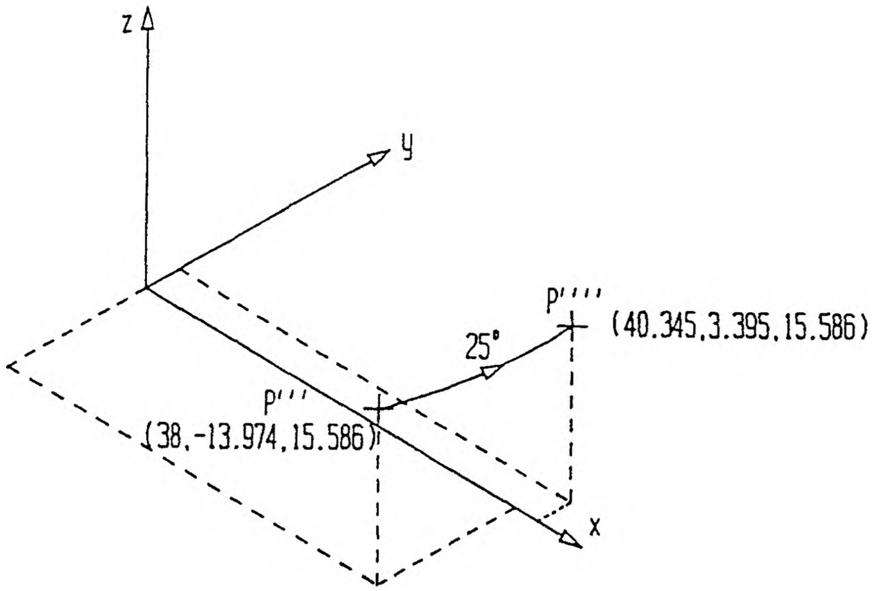


Figure 4.8. z-Rotation with Respect to the Base Coordinate Frame

The other method for consideration of the transformed point begins by performing the first transform on the point vector, generating an intermediate vector. The next transformation is then applied to this new point but now with respect to the coordinate frame resulting from the first transform. The third transformation is applied with respect to the resultant frame of the first two transforms, and the final transformation is applied in turn with respect to the result of the first three transforms. This successive relative coordinate frame transformation scheme is extended from within one link to going from one link to the next to accommodate the motion of the entire manipulator.

The numerical example for this alternate reasoning is more complicated. The sequence begins by rotating the original point about the z-axis. The x- and y-axes are also rotated as can be seen in Figure 4.9.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}' = \begin{bmatrix} \cos 25^\circ & -\sin 25^\circ & 0 & 0 \\ \sin 25^\circ & \cos 25^\circ & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 22 \\ 10 \\ 13 \\ 1 \end{bmatrix} \tag{4.42}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}' = \begin{bmatrix} 15.713 \\ 18.361 \\ 13.000 \\ 1 \end{bmatrix} \tag{4.43}$$

Next, the z-translation is carried out. Note that as the z-axis was not altered by the previous transformation, this translation may be performed immediately. Figure 4.10 shows the changes made to the point and the current coordinate frame.

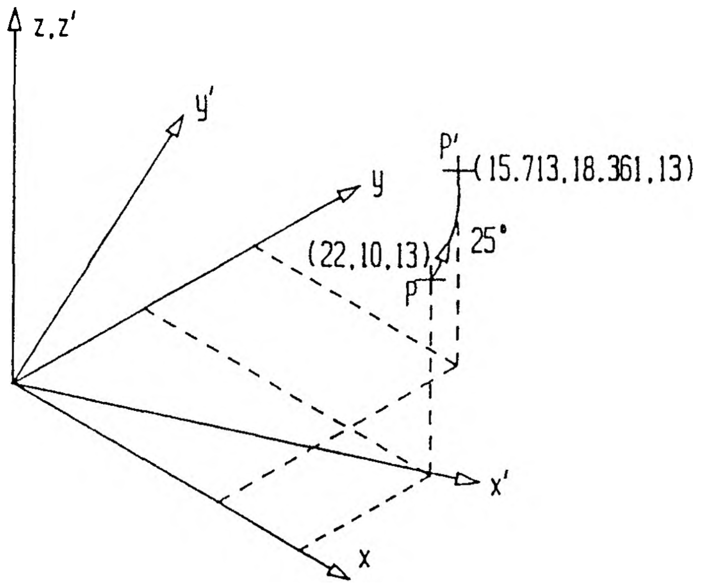


Figure 4.9. z-Rotation with Respect to the Current Coordinate Frame

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}'' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 15.713 \\ 18.361 \\ 13.000 \\ 1 \end{bmatrix} \tag{4.44}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}'' = \begin{bmatrix} 15.713 \\ 18.361 \\ 20.000 \\ 1 \end{bmatrix} \tag{4.45}$$

As can be seen in Figure 4.10, the current x-direction is the intersection of the horizontal plane parallel to the original x-y plane 7 units up the original z-axis and the original x-z plane rotated 25° about the original z-axis. A translation of 16 units in this direction is thus the combination of a translation of $16 \cos 25°$ in the original x-direction and a translation of $16 \sin 25°$ in the original y-direction. The translation is depicted in Figure 4.11.

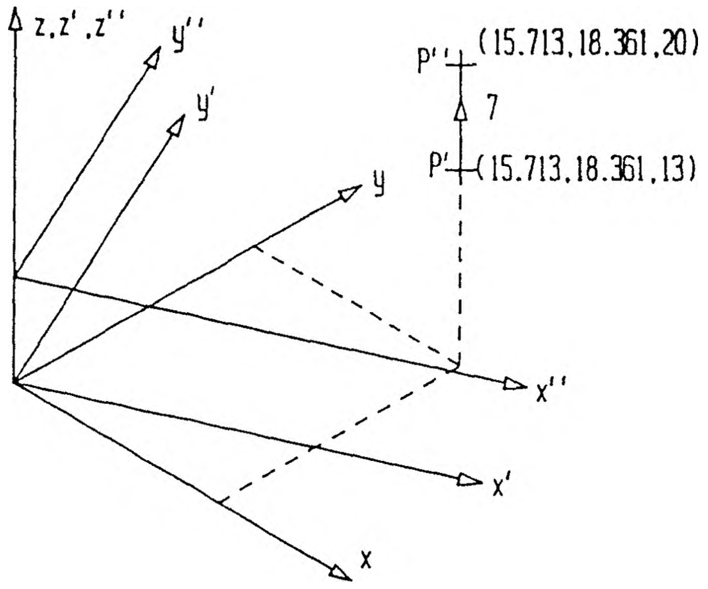


Figure 4.10. z-Translation with Respect to the Current Coordinate Frame

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}''' = \begin{bmatrix} 15.713 + 16\cos 25^\circ \\ 18.361 + 16\sin 25^\circ \\ 20.000 \\ 1 \end{bmatrix} \tag{4.46}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}''' = \begin{bmatrix} 30.214 \\ 25.123 \\ 20.000 \\ 1 \end{bmatrix} \tag{4.47}$$

The final transformation, the x-rotation of p''' about x''', is depicted in Figure 4.12. The numerical determination of the coordinates of p'''' requires several steps. This is because the rotation equations derived earlier only apply to rotations about the origin. Thus the current x-axis (as well as its coordinate frame and p''') must be transformed back to its original direction and position prior to performing the rotation about it.

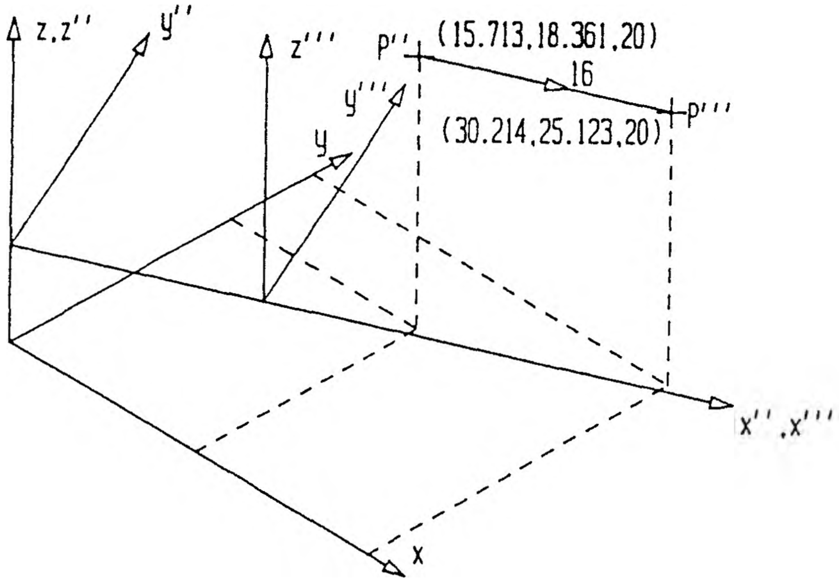


Figure 4.11. x-Translation with Respect to the Current Coordinate Frame

The back-transforms are then undone on the axis, coordinate frame, and p''', resulting in the fourth coordinate frame and p''''.

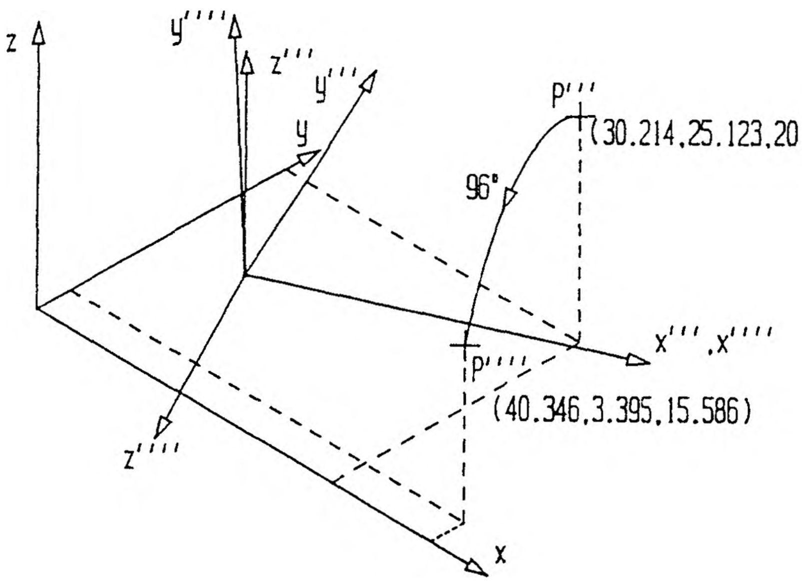


Figure 4.12. x-Rotation with Respect to the Current Coordinate Frame

First, rotate the point (and consequently the current coordinate frame) -25° about the x-axis so that the axis of rotation, x, lies in the base coordinate x-z plane parallel to the base frame x-axis.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{temp1} = \begin{bmatrix} \cos(-25)^\circ & -\sin(-25)^\circ & 0 & 0 \\ \sin(-25)^\circ & \cos(-25)^\circ & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 30.214 \\ 25.123 \\ 20.000 \\ 1 \end{bmatrix} \tag{4.48}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{temp1} = \begin{bmatrix} 38.001 \\ 10.000 \\ 20.000 \\ 1 \end{bmatrix} \tag{4.49}$$

Relative to the current coordinate frame, the point is still $[22,\ 10,\ 13]^T$. The effects of the z- and x-translations of 7 and 16 units, respectively, are still apparent. This point is then rotated about the x-axis by the required 96°.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{temp2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 96^\circ & -\sin 96^\circ & 0 \\ 0 & \sin 96^\circ & \cos 96^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 38.001 - 16 \\ 10.000 \\ 20.000 - 7 \\ 1 \end{bmatrix} \tag{4.50}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{temp2} = \begin{bmatrix} 22.001 \\ -13.974 \\ 8.586 \\ 1 \end{bmatrix} \tag{4.51}$$

The effects of the z- and x-translations are reinstated and the previous z-rotation of -25° is reversed to obtain the final position.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{''''} = \begin{bmatrix} \cos 25^\circ & -\sin 25^\circ & 0 & 0 \\ \sin 25^\circ & \cos 25^\circ & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 22.001 + 16 \\ -13.974 \\ 8.586 + 7 \\ 1 \end{bmatrix} \tag{4.52}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}^{''''} = \begin{bmatrix} 40.346 \\ 3.395 \\ 15.586 \\ 1 \end{bmatrix} \tag{4.53}$$

The results are virtually identical with those obtained in the previous example's Equation (4.41).

The four transformation matrices of Equation (4.33) may be multiplied together to form what is termed the A matrix for link i.

$$A_i = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.54}$$

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.55}$$

3. Assignment of Coordinate Frames. The initial configuration of a joint and the assignment of a coordinate frame to it should be done so that a rotating joint spins

about its z-axis and a sliding joint moves along its z-axis, if at all possible; this will simplify later calculations. The remaining assignment convention requires that all of the coordinate frames' x-axes be aligned in the same direction; this is necessary if the z-rotation is to be the result of a joint movement as the three other transformations above do not provide for any change in x-direction from one coordinate frame to the next. It is also worth noting here that no reference has been made to the y-axis. No provisions have been made for rotation about or translation along a y-axis. The reason here is again that of simplicity; by prohibiting the use of y, subsequent calculations are reduced. Further, since an x-axis and z-axis uniquely determine a y-axis ($z \times x = y$), no y-axis will be pictured as it would only serve to add needless complexity to a figure.

Consider the arm as it rotates in the horizontal plane first. The base coordinate frame was assigned with the z-axis, z_0, in the upward direction with this first rotation in mind so that the first control variable, θ_1, the horizontal rotation of the arm, will be about this axis; see Figure 4.13 for this and the subsequent rotations. The next rotation will be that which moves the arm in the vertical direction, θ_2. The axis of rotation for the angle is z_1 and is assigned with the same base as axis z_0. Further, from the point of view of the base coordinate origin looking along the length of the extended arm, the z_1-axis extends in the right direction. Thus there is no d_1 translation along z_0 to the new base, nor is there any a_1 translation along x_0. The angle α_1 is the movement about x_0 from the z_0 direction to that of z_1, or 90°. Thus the first row of the link parameter table is complete; see Table 4.IV for this and the subsequent rows.

There is no choice for the next rotation of the arm: it is that of the elbow. The arm's elbow can be seen to rotate in the horizontal plane about a vertical axis. The upward direction was selected for this next z-axis, z_2. A translation along x_1, namely $a_2 = 100$ mm, is sufficient to move from the base of the second frame to that of the

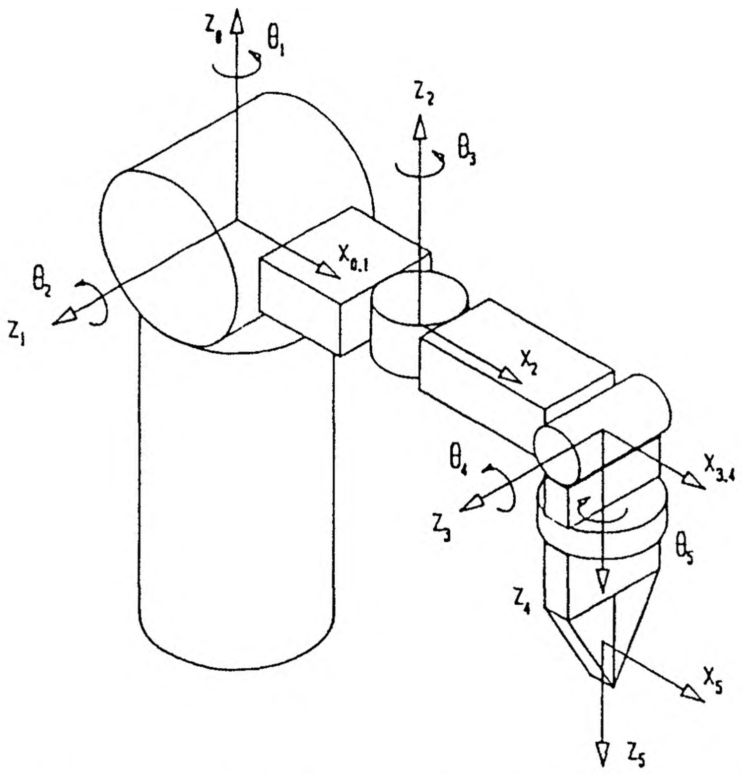


Figure 4.13. Coordinate Frame Assignments to the Robot Arm

third; as no z-translation is necessary, d_2 is zero. The rotation about x_1 which brings z_1 in line with z_2 can be seen to be $-90°$. Thus the assignments for the second row of the link parameter table are made.

The fourth rotation of the arm could be either that causing raising and lowering of the wrist or the rotation of the gripper about its own center line in a drill-type fashion. Since the final offset to the gripper center is in the same direction as the axis of gripper rotation, the former is chosen here so that the offset will take place along z_4 rather than z_5. The base coordinate frame again has the same x-direction but the z-direction now extends horizontally to the right of the joint. A translation along x_2, specifically $a_3 = 100$ mm, suffices for the move from the previous base to the new one; no z-translation is needed so d_3 is zero. The rotation from axis z_2 to z_3 about x_2 can be seen to be 90°. This completes the third row of the link parameter table.

The final rotation is that of the wrist about its own center-line yielding the direct rotation of the gripper. It can now be seen why the initial configuration calls for the gripper to be pointed down. One might be inclined to fully extend the arm with the gripper pointing outward. The rotation of the gripper would then be about the common x-direction. This would in turn complicate subsequent calculations as an α, rather than a θ, rotation would be required. By initially pointing the gripper down and assigning the z-axis z_4 in this direction, the control variable is another θ, θ_5, and effort will be saved later. The transformation from the base used by θ_4 to that of θ_5 requires no translation in either the x- or z-direction as the two frames share the same origin; thus a_4 and d_4 are both zero. The rotation about x_3 from z_3 to z_4 is 90°. This completes the fourth row of the link parameter table.

The final coordinate frame will be dependent on the action of the final control variable, θ_5, and is based at the center point of the gripper closure. There is a z-translation, $d_5 = 100$ mm, along axis z_4 to the center point of the gripper; no x-translation is required so a_5 is zero. The direction for the final coordinate frame's z-axis, z_5, is arbitrary as no further translations or rotations remain; aligning it with z_4 requires no rotation about x_4, so α_5 is zero. This finishes the link parameter table.

Table 4.IV. THE ARMATRON LINK PARAMETER TABLE

Link	Variable	a_i	d_i	α_i
1	θ_1	0	0	+90°
2	θ_2	a_2	0	−90°
3	θ_3	a_3	0	+90°
4	θ_4	0	0	+90°
5	θ_5	0	d_5	0°

4. Generation of Equations. Using the link parameter table, A matrices are created for each link. From Equation (4.55),

$$A_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1\cos\alpha_1 & \sin\theta_1\sin\alpha_1 & a_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1\cos\alpha_1 & -\cos\theta_1\sin\alpha_1 & a_1\sin\theta_1 \\ 0 & \sin\alpha_1 & \cos\alpha_1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.56}$$

Substitutions into Equation (4.56) from the first row of the link parameter table are then made.

$$A_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1\cos 90^\circ & \sin\theta_1\sin 90^\circ & (0)\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1\cos 90^\circ & -\cos\theta_1\sin 90^\circ & (0)\sin\theta_1 \\ 0 & \sin 90^\circ & \cos 90^\circ & (0) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.57}$$

$$A_1 = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1(0) & \sin\theta_1(1) & 0 \\ \sin\theta_1 & \cos\theta_1(0) & -\cos\theta_1(1) & 0 \\ 0 & (1) & (0) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.58}$$

$$A_1 = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ \sin\theta_1 & 0 & -\cos\theta_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.59}$$

The A matrix for link 2 is obtained from Equation (4.55) and the second row of the link parameter table.

$$A_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2\cos(-90^\circ) & \sin\theta_2\sin(-90^\circ) & a_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2\cos(-90^\circ) & -\cos\theta_2\sin(-90^\circ) & a_2\sin\theta_2 \\ 0 & \sin(-90^\circ) & \cos(-90^\circ) & (0) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.60}$$

$$A_2 = \begin{bmatrix} \cos\theta_2 & 0 & -\sin\theta_2 & a_2\cos\theta_2 \\ \sin\theta_2 & 0 & \cos\theta_2 & a_2\sin\theta_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.61}$$

The third row of the link parameter table and Equation (4.55) yield the A matrix for link 3.

$$A_3 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3\cos(90^\circ) & \sin\theta_3\sin(90^\circ) & a_3\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3\cos(90^\circ) & -\cos\theta_3\sin(90^\circ) & a_3\sin\theta_3 \\ 0 & \sin(90^\circ) & \cos(90^\circ) & (0) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.62}$$

$$A_3 = \begin{bmatrix} \cos\theta_3 & 0 & \sin\theta_3 & a_3\cos\theta_3 \\ \sin\theta_3 & 0 & -\cos\theta_3 & a_3\sin\theta_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.63}$$

The A matrix for link 4 is obtained using Equation (4.55) and the fourth row of the link parameter table.

$$A_4 = \begin{bmatrix} \cos\theta_4 & -\sin\theta_4\cos(90^\circ) & \sin\theta_4\sin(90^\circ) & (0)\cos\theta_4 \\ \sin\theta_4 & \cos\theta_4\cos(90^\circ) & -\cos\theta_4\sin(90^\circ) & (0)\sin\theta_4 \\ 0 & \sin(90^\circ) & \cos(90^\circ) & (0) \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.64)$$

$$A_4 = \begin{bmatrix} \cos\theta_4 & 0 & \sin\theta_4 & 0 \\ \sin\theta_4 & 0 & -\cos\theta_4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.65)$$

The fifth and final row of the link parameter table together with Equation (4.55) yields the last A matrix of the Armatron manipulator.

$$A_5 = \begin{bmatrix} \cos\theta_5 & -\sin\theta_5\cos(0^\circ) & \sin\theta_5\sin(0^\circ) & (0)\cos\theta_5 \\ \sin\theta_5 & \cos\theta_5\cos(0^\circ) & -\cos\theta_5\sin(0^\circ) & (0)\sin\theta_5 \\ 0 & \sin(0^\circ) & \cos(0^\circ) & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.66)$$

$$A_5 = \begin{bmatrix} \cos\theta_5 & -\sin\theta_5 & 0 & 0 \\ \sin\theta_5 & \cos\theta_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.67)$$

These five matrices may now be used to transform a point in base coordinates to the position resulting from the five robotic joint settings.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = A_1A_2A_3A_4A_5 \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.68}$$

As described before, one can think of performing the A_1 transformation on the point with respect to the base coordinate frame, then performing the A_2 transformation on the new point with respect to the A_1-transformed coordinate frame, and so on through A_5. This equation will be used to transform the origin to the center point of the gripper closure and it will be used to determine the three vector directions specifying the orientation of the gripper. It is clear from Equation (4.68) that the matrix product $A_1A_2A_3A_4A_5$ must be obtained. The following equations obtain this product by post-multiplying one additional matrix at a time.

$$A_1A_2 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_2 & 0 & -s_2 & a_2c_2 \\ s_2 & 0 & c_2 & a_2s_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.69}$$

$$A_1A_2 = \begin{bmatrix} c_1c_2 & -s_1 & -c_1s_2 & a_2c_1c_2 \\ s_1c_2 & c_1 & -s_1s_2 & a_2s_1c_2 \\ s_2 & 0 & c_2 & a_2s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.70}$$

$$A_1A_2A_3 = \begin{bmatrix} c_1c_2 & -s_1 & -c_1s_2 & a_2c_1c_2 \\ s_1c_2 & c_1 & -s_1s_2 & a_2s_1c_2 \\ s_2 & 0 & c_2 & a_2s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_3 & 0 & s_3 & a_3c_3 \\ s_3 & 0 & -c_3 & a_3s_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.71}$$

$$A_1A_2A_3 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_1s_2 & c_1c_2s_3 + s_1c_3 & a_3c_1c_2c_3 - a_3s_1s_3 + a_2c_1c_2 \\ s_1c_2c_3 + c_1s_3 & -s_1s_2 & s_1c_2s_3 - c_1c_3 & a_3s_1c_2c_3 + a_3c_1s_3 + a_2s_1c_2 \\ s_2c_3 & c_2 & s_2s_3 & a_3s_2c_3 + a_2s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.72}$$

$$A_1A_2A_3A_4 = A_1A_2A_3 \begin{bmatrix} c_4 & 0 & s_4 & 0 \\ s_4 & 0 & -c_4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.73}$$

$$A_1A_2A_3A_4 = \begin{bmatrix} (c_1c_2c_3 - s_1s_3)c_4 - c_1s_2s_4 & c_1c_2s_3 + s_1c_3 & (c_1c_2c_3 - s_1s_3)s_4 + c_1s_2c_4 & a_3(c_1c_2c_3 - s_1s_3) + a_2c_1c_2 \\ (s_1c_2c_3 + c_1s_3)c_4 - s_1s_2s_4 & s_1c_2s_3 - c_1c_3 & (s_1c_2c_3 + c_1s_3)s_4 + s_1s_2c_4 & a_3(s_1c_2c_3 + c_1s_3) + a_2s_1c_2 \\ s_2c_3c_4 + c_2s_4 & s_2s_3 & s_2c_3s_4 - c_2c_4 & (a_3c_3 + a_2)s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.74}$$

$$A_1A_2A_3A_4A_5 = A_1A_2A_3A_4\begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.75}$$

$$A_1A_2A_3A_4A_5 = \begin{bmatrix} ((c_1c_2c_3 - s_1s_3)c_4 - c_1s_2s_4)c_5 + (c_1c_2s_3 + s_1c_3)s_5 \\ ((s_1c_2c_3 + c_1s_3)c_4 - s_1s_2s_4)c_5 + (s_1c_2s_3 - c_1c_3)s_5 \\ (s_2c_3c_4 + c_2s_4)c_5 + s_2s_3s_5 \\ 0 \end{bmatrix.$$

$$\begin{matrix} -((c_1c_2c_3 - s_1s_3)c_4 - c_1s_2s_4)s_5 + (c_1c_2s_3 + s_1c_3)c_5 \\ -((s_1c_2c_3 + c_1s_3)c_4 - s_1s_2s_4)s_5 + (s_1c_2s_3 - c_1c_3)c_5 \\ -(s_2c_3c_4 + c_2s_4)s_5 + s_2s_3c_5 \\ 0 \end{matrix}$$

$$\begin{matrix} (c_1c_2c_3 - s_1s_3)s_4 + c_1s_2c_4 \\ (s_1c_2c_3 + c_1s_3)s_4 + s_1s_2c_4 \\ s_2c_3s_4 - c_2c_4 \\ 0 \end{matrix} \tag{4.76}$$

$$\left.\begin{matrix} d_5((c_1c_2c_3 - s_1s_3)s_4 + c_1s_2c_4) + a_3(c_1c_2c_3 - s_1s_3) + a_2c_1c_2 \\ d_5((s_1c_2c_3 + c_1s_3)s_4 + s_1s_2c_4) + a_3(s_1c_2c_3 + c_1s_3) + a_2s_1c_2 \\ d_5(s_2c_3s_4 - c_2c_4) + (a_3c_3 + a_2)s_2 \\ 1 \end{matrix}\right]$$

With the combined transformation matrix in hand, all that remains is the selection of points to be transformed.

As described earlier, the center point of the gripper closure (referred to as point p) is the base point of the last coordinate frame of reference. The transformation $A_1A_2A_3A_4A_5$, or ${}_0A^5$, translates the base coordinate frame origin to p. Thus

$[0\ \ 0\ \ 0\ \ 1]^T$ shall have the transform applied to it to obtain the current gripper postion, which is referred to as $[p_x\ \ p_y\ \ p_z\ \ 1]^T$. See Figure 4.14.

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = {}_0A^5 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.77}$$

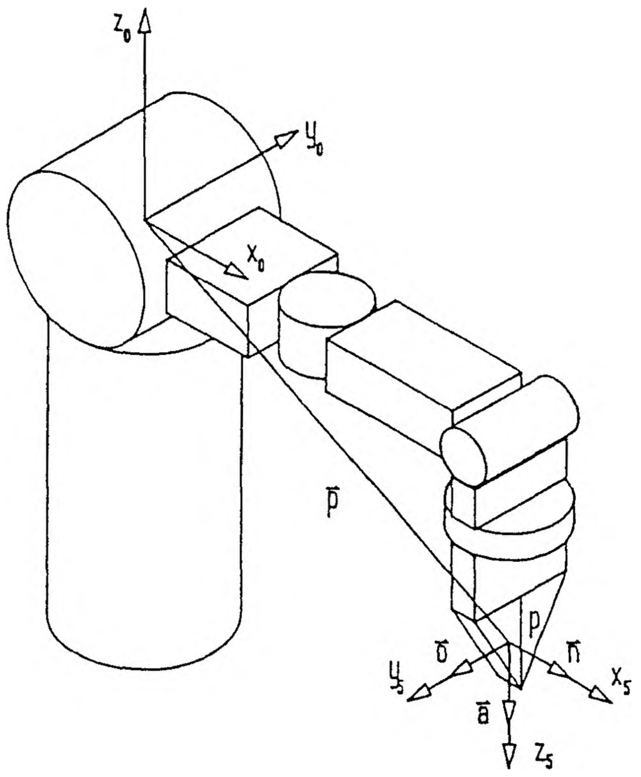


Figure 4.14. Position Point and Orientation Vectors

The vector $\vec{p}$ is that extending from the base coordinate frame origin to point p. The components of $\vec{p}$ are then found by subtraction of the components of the end point from the start point.

$$\vec{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.78}$$

$$\vec{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 0 \end{bmatrix} \tag{4.79}$$

Three vectors with which the orientation of the gripper may be described will now be obtained under the transformation; see vectors $\vec{n}$, $\vec{o}$, and $\vec{a}$ in Figure 4.14. First consider a unit vector in the direction of the base frame's z-axis. When the transform is applied to a point along this vector, a point along the final z-axis, z_5, is obtained. This axis has its origin at the gripper center and lies along the gripper center line; the ${}_0A^5$ transformation of a point along the base z-axis thus yields a point along the gripper center line. The unit vector in this direction is termed the approach vector, $\vec{a}$. Transformation of the base coordinate point $[0 \;\; 0 \;\; 1 \;\; 1]^T$ would yield the desired point on the z_5-axis, but the vector itself (i.e., a direction) would be of more use. The approach vector is obtained by subtracting the gripper center from the transformed $[0 \;\; 0 \;\; 1 \;\; 1]^T$.

$$\vec{a} = {}_0A^5 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{4.80}$$

Substituting from Equation (4.77) into Equation (4.80) yields a simplified result for the approach vector.

$$\vec{a} = {}_0A^5 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} - {}_0A^5 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4.81}$$

$$\vec{a} = {}_0A^5 \left[\begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right] \tag{4.82}$$

$$\vec{a} = {}_0A^5 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{4.83}$$

Next consider a unit vector in the direction of the base frame's x-axis. A point along the final x-axis x_5 is obtained when the ${}_0A^5$ transform is applied. The unit vector in this direction is designated the normal vector $\vec{n}$. Transformation of the point $[1 \ \ 0 \ \ 0 \ \ 1]^T$ would yield the desired endpoint, but transformation of $[1 \ \ 0 \ \ 0 \ \ 0]^T$ yields the vector components of the n vector as was demonstrated for the approach vector.

$$\vec{n} = {}_0A^5 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.84}$$

Lastly, a unit vector in the direction of the base frame's y-axis is transformed. The unit vector in the direction of the final y-axis, y_5, is called the orientation vector, $\vec{o}$. In the same manner as that used for the vectors $\vec{a}$ and $\vec{n}$, transformation of $[0 \quad 1 \quad 0 \quad 0]^T$ yields the vector components of $\vec{o}$.

$$\vec{o} = {}_0A^5 \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{4.85}$$

Equations (4.77), (4.83), (4.84), and (4.85) may now be combined into a single matrix equation.

$$\begin{bmatrix} p & \vec{a} & \vec{n} & \vec{o} \end{bmatrix} = {}_0A^5 \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{4.86}$$

Observing that the four columns of the rightmost matrix are also columns of the identity matrix, the equation may be rearranged to take advantage of this.

$$\begin{bmatrix}\vec{n} & \vec{o} & \vec{a} & p\end{bmatrix} = {}_0A^5 \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.87}$$

$$\begin{bmatrix}\vec{n} & \vec{o} & \vec{a} & p\end{bmatrix} = {}_0A^5 \tag{4.88}$$

Expanding the vectors and point into their components,

$$\begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}_0A^5 \tag{4.89}$$

Thus, the elements of the matrix product ${}_0A^5$ are themselves components of the desired position and orientation vectors.

Procedure **`noap_matrix`**, mentioned in Chapters 1 and 3, calculates and displays the components of the vectors of the vectors $\vec{n}$, $\vec{o}$, $\vec{a}$, and $\vec{p}$, respectively, as well as the joint variables. The equations for $\vec{n}$ are obtained from Equation (4.76).

$$n_x = ((c_1c_2c_3 - s_1s_3)c_4 - c_1s_2s_4)c_5 + (c_1c_2s_3 + s_1c_3)s_5 \tag{4.90}$$

$$n_y = ((s_1c_2c_3 + c_1s_3)c_4 - s_1s_2s_4)c_5 + (s_1c_2s_3 - c_1c_3)s_5 \tag{4.91}$$

$$n_z = (s_2c_3c_4 + c_2s_4)c_5 + s_2s_3s_5 \tag{4.92}$$

The component equations for $\vec{o}$ are:

$$o_x = -((c_1c_2c_3 - s_1s_3)c_4 - c_1s_2s_4)s_5 + (c_1c_2s_3 + s_1c_3)c_5 \tag{4.93}$$

$$o_y = -((s_1c_2c_3 + c_1s_3)c_4 - s_1s_2s_4)s_5 + (s_1c_2s_3 - c_1c_3)c_5 \tag{4.94}$$

$$o_z = -(s_2c_3c_4 + c_2s_4)s_5 + s_2s_3c_5 \tag{4.95}$$

Vector $\vec{a}$ components are defined by the following equations:

$$a_x = (c_1c_2c_3 - s_1s_3)s_4 + c_1s_2c_4 \tag{4.96}$$

$$a_y = (s_1c_2c_3 + c_1s_3)s_4 + s_1s_2c_4 \tag{4.97}$$

$$a_z = s_2c_3s_4 - c_2c_4 \tag{4.98}$$

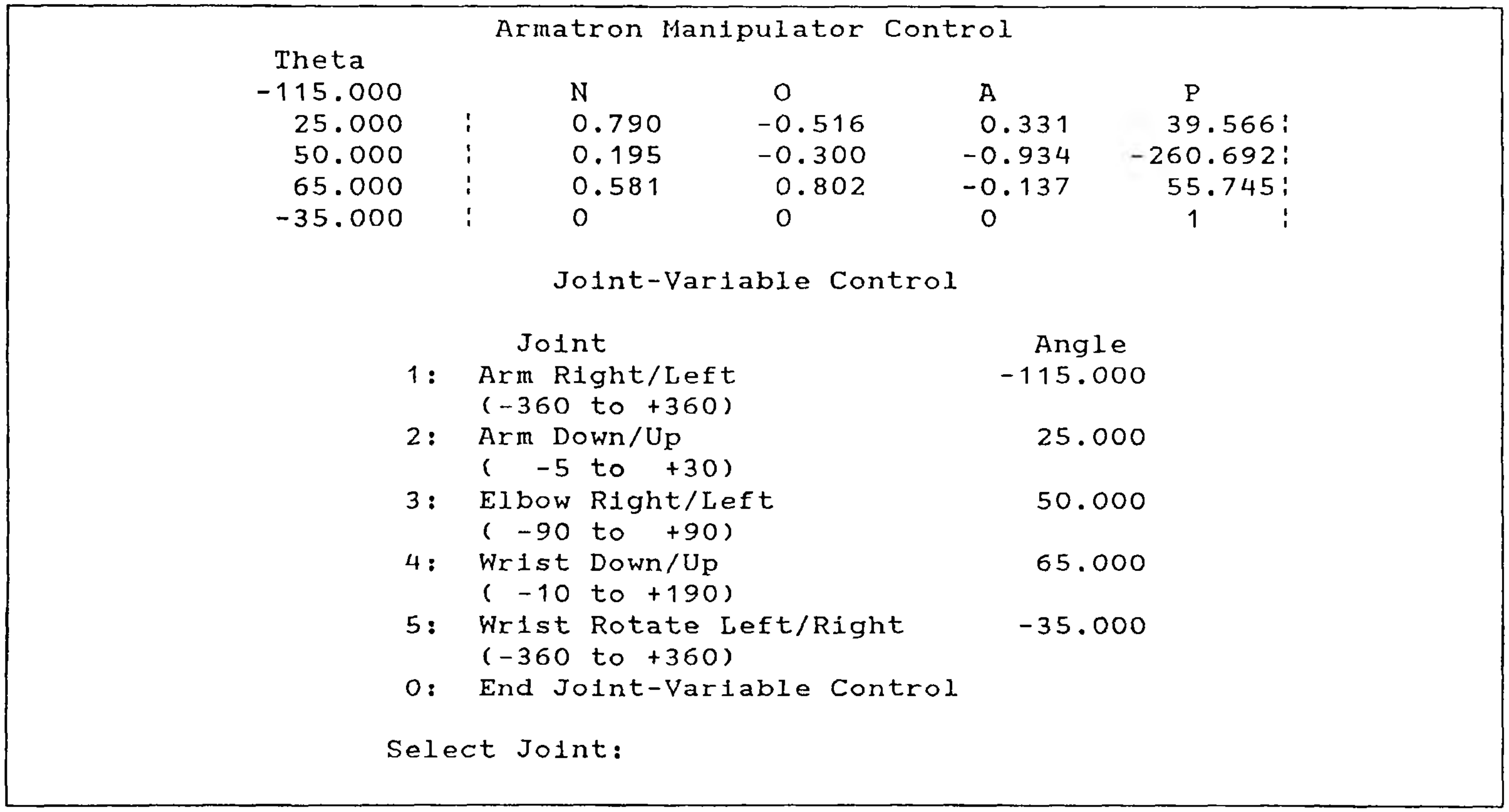
Lastly, the component equations for point p are:

$$p_x = d_5((c_1c_2c_3 - s_1s_3)s_4 + c_1s_2c_4) + a_3(c_1c_2c_3 - s_1s_3) + a_2c_1c_2 \tag{4.99}$$

$$p_y = d_5((s_1c_2c_3 + c_1s_3)s_4 + s_1s_2c_4) + a_3(s_1c_2c_3 + c_1s_3) + a_2s_1c_2 \tag{4.100}$$

$$p_z = d_5(s_2c_3s_4 - c_2c_4) + (a_3c_3 + a_2)s_2 \tag{4.101}$$

5. Numerical Example. As an example, consider the set of control variable values $(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5) = (-115^\circ, 25^\circ, 50^\circ, 65^\circ, -35^\circ)$. After these five values have been input as settings for the respective joints, the display in Figure 4.15 is produced.



```
                    Armatron Manipulator Control
  Theta
 -115.000          N          O          A          P
   25.000  |     0.790     -0.516      0.331      39.566|
   50.000  |     0.195     -0.300     -0.934    -260.692|
   65.000  |     0.581      0.802     -0.137      55.745|
  -35.000  |     0              0          0          1      |

                      Joint-Variable Control

                    Joint                          Angle
           1:   Arm Right/Left                 -115.000
                (-360 to +360)
           2:   Arm Down/Up                      25.000
                (  -5 to  +30)
           3:   Elbow Right/Left                 50.000
                ( -90 to  +90)
           4:   Wrist Down/Up                    65.000
                ( -10 to +190)
           5:   Wrist Rotate Left/Right         -35.000
                (-360 to +360)
           0:   End Joint-Variable Control

          Select Joint:
```

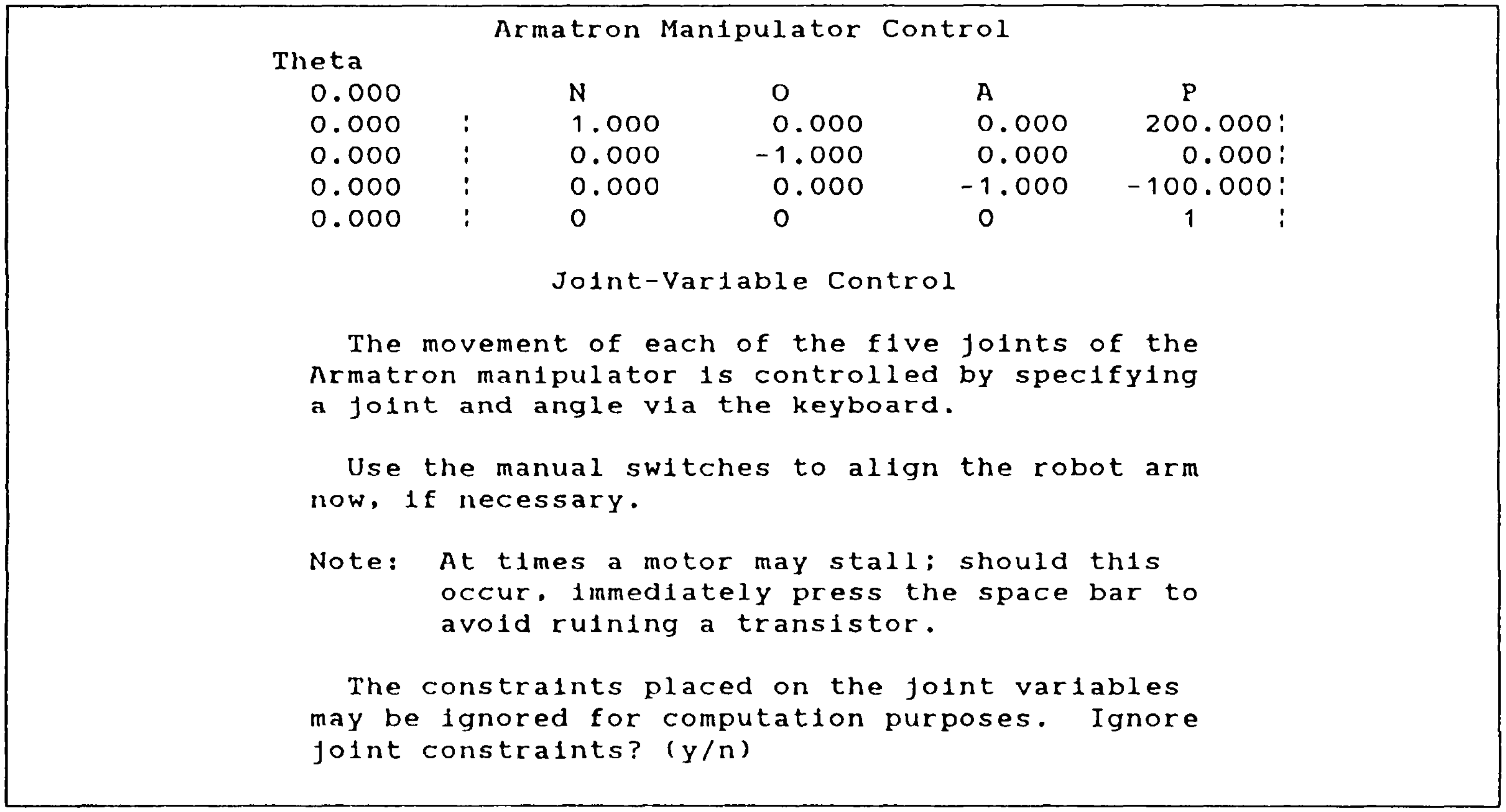
Figure 4.15. Joint Variable Control Display

D. THE CONTROLLING PROCEDURE

Procedure `joint_variable_control` begins by displaying the introductory screen of Figure 4.16. The option is given at that point of using the actual joint limitations for the manipulator joints or bypassing these so that this portion of the program may be used for computation purposes only. Orientation and position matrices may be generated for any values of joint variables if the constraints are bypassed. The next chapter develops a program which takes as input a position-orientation matrix and determines possible solution sets. With the joint constraints of this chapter's procedures ignored, they may be used to verify those joint solutions, whether they are attainable or not. The position-orientation matrix determined for the numerical example in the preceding section will be used as the input for the numerical example of the next chapter.

```
dsply_joint_variable_introduction ( );
locate (23, 42);
ignore = toupper(getch( ));
lputch (23, 42, ignore);
constraints (ignore, theta_min, theta_max);
wait_then_erase (9);
dsply_joint_variables (jv_rows, &jv_col);
for (i = 1; i <= 5; i++)
   lcprintf (jv_rows[i], jv_col, theta[i]);
process_requests (theta, noap, theta_min, theta_max,
                  jv_rows, jv_col, noap_row, noap_cols);
erase_prompt (23);
locate (23, 0);
mcputs (24, 23, "Joint-Variable Control Terminated");
wait_then_erase (8);
```

Subsequently, the working display of the procedure is generated and control passes to the request processing procedure, where it remains until joint variable control is to be terminated. The documented listing for the procedures associated with the joint variable control portion of the overall program may be found in Appendix D.



```
                  Armatron Manipulator Control
       Theta
         0.000          N          O          A          P
         0.000  :     1.000      0.000      0.000    200.000:
         0.000  :     0.000     -1.000      0.000      0.000:
         0.000  :     0.000      0.000     -1.000   -100.000:
         0.000  :     0             0          0          1     :

                     Joint-Variable Control

          The movement of each of the five joints of the
        Armatron manipulator is controlled by specifying
        a joint and angle via the keyboard.

          Use the manual switches to align the robot arm
        now, if necessary.

        Note:  At times a motor may stall; should this
               occur, immediately press the space bar to
               avoid ruining a transistor.

          The constraints placed on the joint variables
        may be ignored for computation purposes.  Ignore
        joint constraints? (y/n)
```

Figure 4.16. Joint Variable Control Introduction

V. POSITION AND ORIENTATION CONTROL

The previous chapter derived the center position of the gripper closure as the components of point p and the triple of unit orientation vectors $\vec{n}$, $\vec{o}$, and $\vec{a}$ which describe the orientation of the gripper. The topic of this chapter is the programmed solution of the inverse of this problem. The vector components of $\vec{n}$, $\vec{o}$, and $\vec{a}$ will be given as those of the desired orientation of the gripper, while the point p will specify the desired position. The program will be developed in a step-by-step manner as the problem is solved.

A. SPECIFICATION OF POSITION AND ORIENTATION VECTORS

The first step toward the solution of the stated problem is to provide for the specification of the components of vectors $\vec{n}$, $\vec{o}$, and $\vec{a}$ and the coordinates of point p. The following section details the derivation of equations concerning the orientation vectors and position. Next, a numerical example is employed to demonstrate use of the equations. This is followed by an explanation of the structure of the program code to be used. Finally, the program is demonstrated by example using the same inputs as the numerical example.

1. Derivation of Equations. There exist several conditions concerning the orientation and position of the coordinate frame at the end of the Armatron manipulator which must be met; this section shall develop corresponding equations which must be satisfied by any desired position and orientation before a solution can be attempted. To begin with, each of the orientation vectors $\vec{n}$, $\vec{o}$, and $\vec{a}$ must have a magnitude of one. The magnitude of the normal vector, for example, is found as follows:

$$|\vec{n}| = \sqrt{n_x^2 + n_y^2 + n_z^2} \tag{5.1}$$

The condition may be stated in different ways, two of which follow for the normal vector:

$$|\vec{n}| = 1 \tag{5.2}$$

$$|\ |\vec{n}| - 1\ | < \varepsilon \tag{5.3}$$

In the second expression, ε is some small tolerance which the imprecision resulting from the use of a finite number of decimal positions in the calculations should not exceed; this condition is suitable for use by a computer.

Another condition that must be satisfied by the orientation vectors is that they form a right hand triple, as do the base coordinate frame *x*-, *y*-, and *z*-axes. This condition may be expressed in terms of the cross product. The base frame *x*-axis when crossed with the *y*-axis results in the *z*-axis. Correspondingly, vector $\vec{n}$ crossed with vector $\vec{o}$ should result in vector $\vec{a}$.

$$\vec{n} \times \vec{o} = \begin{bmatrix} n_y o_z - o_y n_z \\ n_z o_x - n_x o_z \\ n_x o_y - n_y o_x \end{bmatrix} \tag{5.4}$$

$$\vec{n} \times \vec{o} = \vec{a} \tag{5.5}$$

$$(\vec{n} \times \vec{o}) - \vec{a} < \begin{bmatrix} \varepsilon \\ \varepsilon \\ \varepsilon \end{bmatrix} \tag{5.6}$$

When the magnitude and cross product conditions are met by a set of orientation vectors, a solution may be attempted for achieving them; they may yet be unattainable, however.

Different facts are known about the position p. (Recall from Chapter 4 that as the position vector $\vec{p}$ was seen to extend from the base coordinate frame origin to point p, the components of $\vec{p}$ are identically the coordinates of p, namely p_x, p_y, and p_z. These entities will be referred to as either point coordinates or vector components depending upon the current context throughout this chapter.) What can be stated concerns the magnitude of vector $\vec{p}$.

$$|\vec{p}\,| = \sqrt{p_x^2 + p_y^2 + p_z^2} \tag{5.7}$$

The magnitude changes over a wide range of values; the space reachable by the arm is termed its envelope. Desa and Roth [Desa85] and Duffy [Duff80] discuss the envelope or workspace of manipulators. Desa and Roth note that not neccessarily all of the positions within an envelope can be approached from an arbitrary direction; they differentiate between primary and secondary workspaces based on this criterion. Inspection of Figure 5.1 shows that the magnitude actually depends on only control variables θ_3 and θ_4. Of the remaining variables, θ_1 and θ_2 affect the vector direction only, while variable θ_5 has no effect on magnitude or direction. It is clear to see that the magnitude of the position vector takes on its maximum value when the arm is fully extended.

$$|\vec{p}\,|_{\max} = a_2 + a_3 + d_5 \tag{5.8}$$

The arm offsets $a_2 = 100$ mm, $a_3 = 100$ mm, and $d_5 = 100$ mm yield a maximum obtainable magnitude of 300 mm.

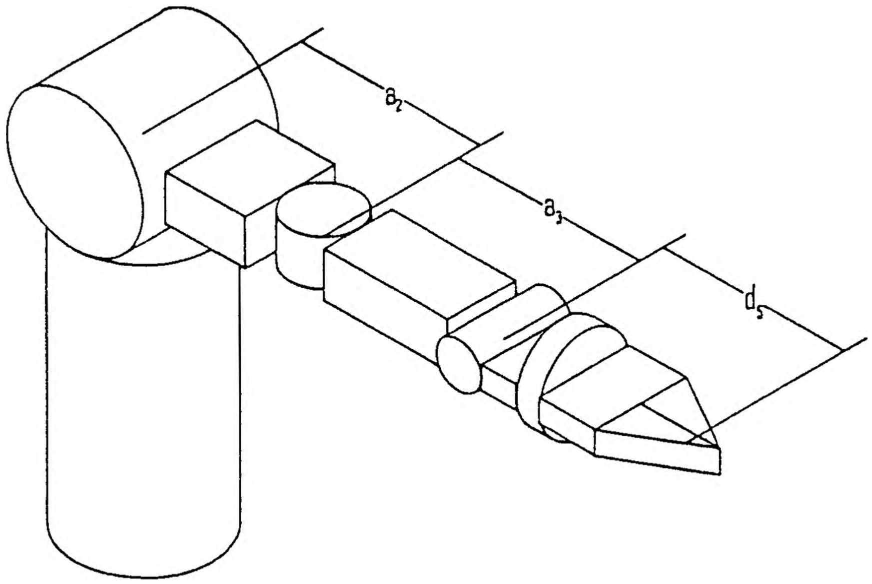


Figure 5.1. Maximum Magnitude Attainable by the Position Vector

Taking into account the physical limitations of the relevant joints, the magnitude of the position vector would take on a minimum value when the conditions in Figure 5.2 are in effect. Joints 3 and 4 can be seen to be at the limits of their arcs of movement, bringing the gripper center as close as possible to the origin. As the wrist is capable of 100° of vertical movement from the horizontal, the wrist will form an 80° angle with the arm. The minimum magnitude for vector $\vec{p}$ is then obtained by a series of geometric calculations. From the figure,

$$l_1 = d_5 \cos 80^\circ \tag{5.9}$$

$$l_2 = d_5 \sin 80^\circ \tag{5.10}$$

$$l_3 = a_3 - l_1 \tag{5.11}$$

$$l_4 = \sqrt{a_2^{\,2} + l_3^{\,2}} \tag{5.12}$$

$$|\vec{p}\,|_{\min} = \sqrt{l_2^{\,2} + l_4^{\,2}} \tag{5.13}$$

The calculations with the arm offsets yield a minimum magnitude of 162.871 mm.

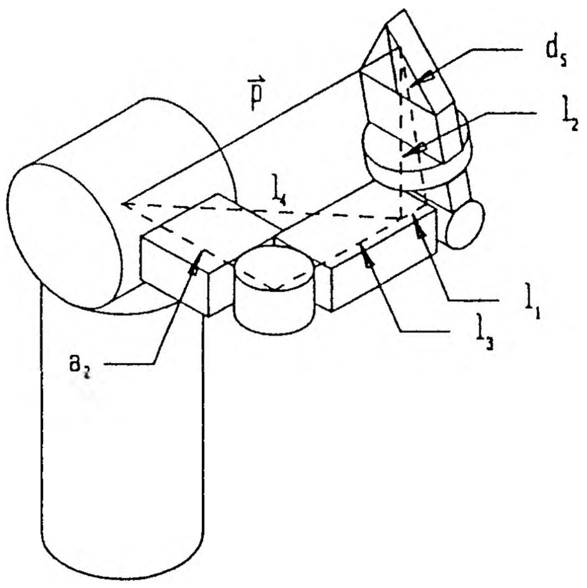


Figure 5.2. Minimum Magnitude Attainable by the Position Vector

The solution process of this chapter shall not be solely concerned with the restraints placed on the robot joints, however. The primary purpose here is to determine solutions for position-orientation specifications. To this end, the joint constraints will be ignored upon entry of the position-orientation matrix; in fact, they shall be ignored until actual solutions are determined. At that point, no move will be attempted if the constraints are violated. Approaching the situation in this fashion will allow for the solution of a much wider range of inputs. A second examination of Figure 5.2 will show that if joint 4 is allowed to fold the wrist back upon the arm, the gripper center would still be a distance of a_2 from the origin. Alternatively, if joint 3 is allowed to fold the arm back on itself at the elbow, the gripper center would be a distance of d_5 from the origin. As a_2, a_3, and d_5 are all equal at 100 mm, there is no way that the joints can be positioned to reduce the position vector magnitude any further. Thus, either a_2 or d_5 may be used as the vector magnitude minimum.

$$|\vec{p}\,|_{\min} = a_2 \tag{5.14}$$

2. Numerical Example. Consider the position-orientation matrix given by the following equation.

$$\begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.790 & -0.516 & 0.331 & 39.566 \\ 0.195 & -0.300 & -0.934 & -260.692 \\ 0.581 & 0.802 & -0.137 & 55.745 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.15}$$

Recall that this matrix was generated as an example by the program of the preceding chapter. The magnitude of vector $\vec{n}$ is found using Equation (5.1).

$$|\vec{n}| = \sqrt{n_x^2 + n_y^2 + n_z^2} \tag{5.16}$$

$$|\vec{n}| = \sqrt{(0.790)^2 + (0.195)^2 + (0.581)^2} = 1.000 \tag{5.17}$$

The magnitudes of vectors $\vec{o}$ and $\vec{a}$ are found similarly.

$$|\vec{o}| = \sqrt{(-0.516)^2 + (-0.300)^2 + (0.802)^2} = 0.999 \tag{5.18}$$

$$|\vec{a}| = \sqrt{(0.331)^2 + (-0.934)^2 + (-0.137)^2} = 1.001 \tag{5.19}$$

With precision limited to three decimal places, a value for ε of 0.001 would be a minimum; something on the order of 0.01 might be more appropriate.

The cross product of vectors $\vec{n}$ and $\vec{o}$ would then be compared to vector $\vec{a}$ using the expression of Equation (5.4).

$$(\vec{n} \times \vec{o}) - \vec{a} = \begin{bmatrix} n_y o_z - o_y n_z \\ n_z o_x - n_x o_z \\ n_x o_y - n_y o_x \end{bmatrix} - \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \tag{5.20}$$

$$(\vec{n} \times \vec{o}) - \vec{a} = \begin{bmatrix} (0.195)(0.802) - (-0.300)(0.581) \\ (0.581)(-0.516) - (0.790)(0.802) \\ (0.790)(-0.300) - (0.195)(-0.516) \end{bmatrix} - \begin{bmatrix} 0.331 \\ < -0.934 \\ -0.137 \end{bmatrix} = \begin{bmatrix} 0.000 \\ 0.001 \\ 0.001 \end{bmatrix} \quad (5.21)$$

Since a solution based on the orientation vectors is feasible, the examination continues with the desired position of the gripper center. The magnitude of the vector was given by Equation (5.7).

$$|\vec{p}\,| = \sqrt{p_x^2 + p_y^2 + p_z^2} \quad (5.22)$$

$$|\vec{p}\,| = \sqrt{(39.566)^2 + (-260.692)^2 + (55.745)^2} = 269.506 \quad (5.23)$$

This value is between the maximum of 300 established in Equation (5.8) and the minimum of 100 established in Equation (5.14).

3. Program Structure. The orientation and position matrix is obtained from the keyboard under direction of procedure `get_noap`. It is at this level that the cross product of orientation vectors is examined; the iteration of the procedure will not be terminated until a valid cross product is obtained.

```
init_names (names);
do
   {
   for (i = 0; i <= 2; i++)
      get_orientation_vector (i, names, noap, row, cols);
   n_cross_o[0] = noap[0][1]*noap[1][2] - noap[0][2]*noap[1][1];
   n_cross_o[1] = noap[0][2]*noap[1][0] - noap[0][0]*noap[1][2];
   n_cross_o[2] = noap[0][0]*noap[1][1] - noap[0][1]*noap[1][0];
   difference = fabs(magnitude (n_cross_o) - magnitude (noap[2]));
   if ( difference > tolerance )
      {
      lcputs (23, 20, "N x O does not equal A; ");
      cputs ("Re-enter Vectors N, O, and A");
      wait_then_erase (23);
      }
   }
   while ( difference > tolerance );
get_position_vector (names, noap, row, cols);
```

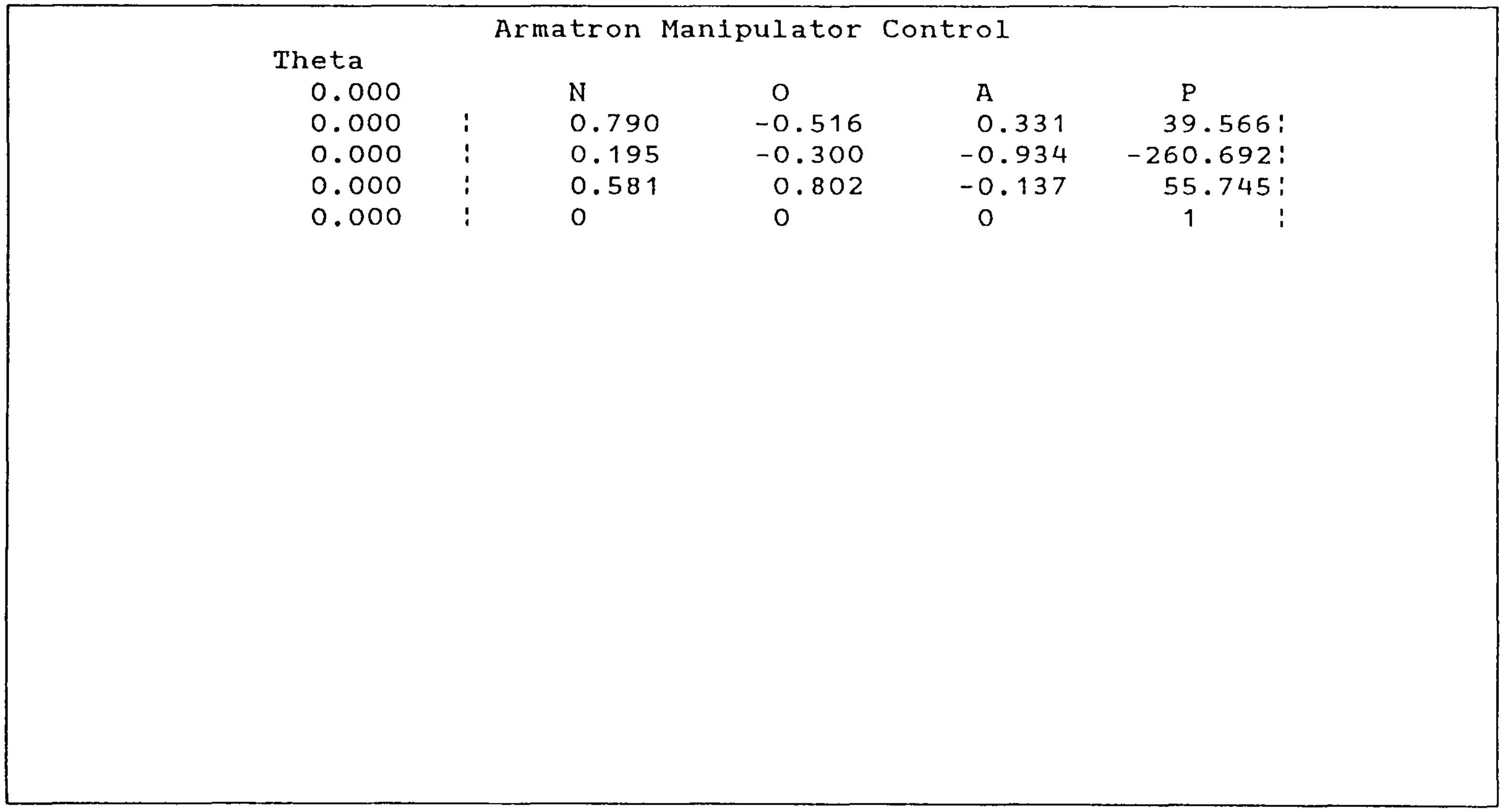
The individual orientation vectors are obtained by procedure `get_orientation_vector`. This procedure will not be exited until the magnitude of the specified vector is sufficiently close to one.

```
do
   {
   for (j = 0; j <= 2; j++)
      {
      prompt_input_noap (names[i][j], &noap[i][j]);
      erase_prompt (23);
      lcprintf (row+j, cols[i], noap[i][j]);
      }
   difference = fabs(1 - magnitude (noap[i]));
   if ( difference > tolerance )
      {
      lcputs (23, 20, "Vector Magnitude does not equal 1; ");
      cputs ("Re-enter Vector ");
      putch (names[i][1][0]);
      wait_then_erase (23);
      }
   }
   while ( difference > tolerance );
```

The position vector components are input under control of procedure `get_position_vector`. The magnitude of the specified vector must fall within the maximum and minimum values derived for the position vector before control returns to the invoking procedure.

```
do
   {
   for (j = 0; j <= 2; j++)
      {
      prompt_input_noap (names[3][j], &noap[3][j]);
      erase_prompt (23);
      lcprintf (row+j, cols[3], noap[3][j]);
      }
   mag_p = magnitude (noap[3]);
   if ( (mag_p > a2+a3+d5) ¦ (mag_p < a2) )
      {
      lcputs (23, 20, "Specified position is outside of the arm ");
      cputs ("envelope; re-enter vector P");
      wait_then_erase (23);
      }
   }
   while ( (mag_p > a2+a3+d5) ¦ (mag_p < a2) );
```

4. Program Example. Execution of procedure `get_noap` generates the display of the information in Figure 5.3; this same display shall be added to by the program examples of the solution process in the following sections.



```
                    Armatron Manipulator Control
      Theta
        0.000               N          O          A          P
        0.000    ¦      0.790     -0.516      0.331      39.566¦
        0.000    ¦      0.195     -0.300     -0.934    -260.692¦
        0.000    ¦      0.581      0.802     -0.137      55.745¦
        0.000    ¦      0          0          0           1    ¦
```

Figure 5.3. Display for Completed Position-Orientation Matrix

B. SOLUTION APPROACHES

The previous chapter detailed the derivation of the following twelve equations:

$$n_x = ((c_1c_2c_3 - s_1s_3)c_4 - c_1s_2s_4)c_5 + (c_1c_2s_3 + s_1c_3)s_5 \tag{5.24}$$

$$n_y = ((s_1c_2c_3 + c_1s_3)c_4 - s_1s_2s_4)c_5 + (s_1c_2s_3 - c_1c_3)s_5 \tag{5.25}$$

$$n_z = (s_2c_3c_4 + c_2s_4)c_5 + s_2s_3s_5 \tag{5.26}$$

$$o_x = -((c_1c_2c_3 - s_1s_3)c_4 - c_1s_2s_4)s_5 + (c_1c_2s_3 + s_1c_3)c_5 \tag{5.27}$$

$$o_y = -((s_1c_2c_3 + c_1s_3)c_4 - s_1s_2s_4)s_5 + (s_1c_2s_3 - c_1c_3)c_5 \tag{5.28}$$

$$o_z = -(s_2c_3c_4 + c_2s_4)s_5 + s_2s_3c_5 \tag{5.29}$$

$$a_x = (c_1c_2c_3 - s_1s_3)s_4 + c_1s_2c_4 \tag{5.30}$$

$$a_y = (s_1c_2c_3 + c_1s_3)s_4 + s_1s_2c_4 \tag{5.31}$$

$$a_z = s_2c_3s_4 - c_2c_4 \tag{5.32}$$

$$p_x = d_5((c_1c_2c_3 - s_1s_3)s_4 + c_1s_2c_4) + a_3(c_1c_2c_3 - s_1s_3) + a_2c_1c_2 \tag{5.33}$$

$$p_y = d_5((s_1c_2c_3 + c_1s_3)s_4 + s_1s_2c_4) + a_3(s_1c_2c_3 + c_1s_3) + a_2s_1c_2 \tag{5.34}$$

$$p_z = d_5(s_2c_3s_4 - c_2c_4) + (a_3c_3 + a_2)s_2 \tag{5.35}$$

This chapter assumes that the twelve left-hand sides of these equations are specified while the angles of the right-hand sides are unknown. An immediate attempt to solve for the five control variables would be difficult and tedious, although this approach has been used.

An alternative approach greatly simplifies the trigonometry and algebra required for the control variable solutions. The idea is to prevent the equations necessary for the solutions from ever becoming as complicated as Equations (5.24) through (5.35). The technique first makes use of the fact that the translation of the entire robot arm, represented by the position vector, is the sum of two translations, the translation of the arm up to the wrist, links 1, 2, and 3, and the translation of the wrist proper, links 4 and 5. Now since the translation due to the wrist has a fixed magnitude, d_5, and a specified direction, that of the approach vector, this second translation vector may be determined immediately. This in turn produces the arm vector by subtraction of the wrist vector from the entire arm vector. Next, the product of the A matrices for links 1, 2, and 3 will yield the position of the arm preceding the wrist as its fourth column. Equating these elements with those obtained from the vector subtraction above yields three equations for the first three control variables. Finally, with the solutions for these variables in hand, the matrix equation relating the product of the A matrices to the specified vectors $\bar{n}$, $\bar{o}$, $\bar{a}$, and $\bar{p}$ is used; the matrices for links 4 and 5 are isolated using matrix inverses. Equations for the solutions of control variables 4 and 5 are then obtained by equating matrix elements. The following three sections will detail each of these steps for the Armatron arm in the four parts used in the previous section: derivation of equations, numerical example, program structure, and program example.

C. COMPONENTS OF THE ARM VECTOR

1. Derivation of Equations. The first step in the solution process is the determination of the components of the arm vector. Consider the robot arm as depicted in Figure 5.4. Note that the position vector, $\bar{p}$, extends from the origin of the base coordinate frame to the gripper center. Now consider the decomposition of the

position vector described in the figure. Clearly, the vector addition of the two components, $\vec{p}_a$ and $\vec{p}_w$, results in the total translation of the arm.

$$\vec{p} = \vec{p}_a + \vec{p}_w \tag{5.36}$$

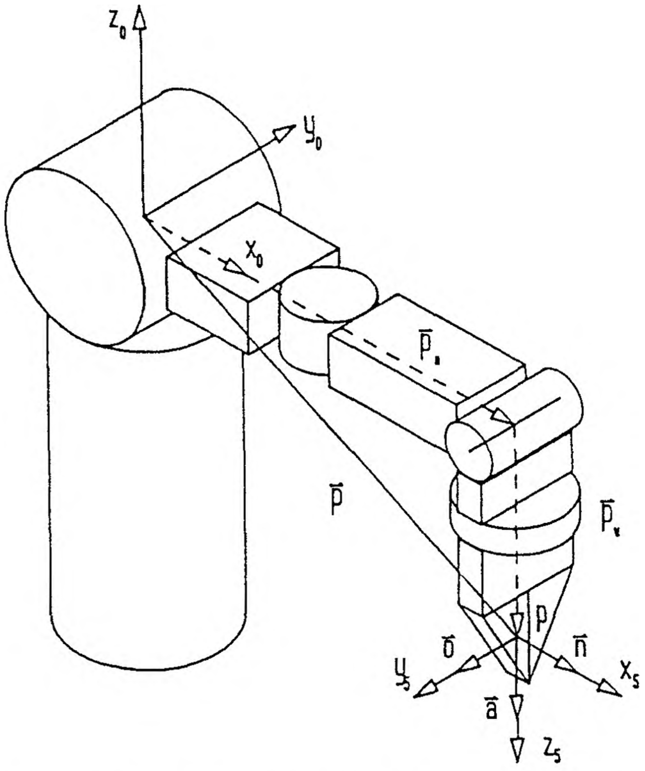


Figure 5.4. Decomposition of the Position Vector

The vector $\vec{p}_a$ represents the translation due to the arm proper, links 1, 2, and 3, while the vector $\vec{p}_w$ denotes the translation of the wrist, links 4 and 5. The components of the vector $\vec{p}$ are known. The components of the vector $\vec{p}_w$ may now be determined since the magnitude of the vector is simply d_5 and the direction is that of the approach vector, whose components are given. At this point, vector $\vec{p}_a$ may be determined from Equation (5.36).

$$\vec{p}_a = \vec{p} - \vec{p}_w \tag{5.37}$$

The translation of $\vec{p_a}$ is due to the two constant arm offsets a_2 and a_3 and the action of the arm control variables θ_1, θ_2, and θ_3. The next section will then detail the derivation of θ_1, θ_2, and θ_3 from the three arm translation components, p_{ax}, p_{ay}, and p_{az}.

Returning to the derivation of the wrist vector components, consider Figure 5.4 again. Note first that vector $\vec{p_w}$ lies in the same direction as vector $\vec{a}$. Vector $\vec{a}$ is itself a unit vector with known components. The components of $\vec{p_w}$ may be found by scaling the components of $\vec{a}$ by $\vec{p_w}$'s magnitude, which is just d_5.

$$p_{w_x} = d_5 a_x \tag{5.38}$$

$$p_{w_y} = d_5 a_y \tag{5.39}$$

$$p_{w_z} = d_5 a_z \tag{5.40}$$

With the components of $\vec{p_w}$ known, the vectors of Equation (5.37) are broken into components to find the constituents of the arm vector.

$$p_{a_x} = p_x - p_{w_x} \tag{5.41}$$

$$p_{a_y} = p_y - p_{w_y} \tag{5.42}$$

$$p_{a_z} = p_z - p_{w_z} \tag{5.43}$$

An examination must now be made to insure that this position will lead to a solution. The coordinates of the gripper center were examined upon input to insure that the desired position was feasible. At that time, however, no relationship had been established between the specified position and the orientation vectors, specifically the approach vector. The wrist vector $\vec{p_w}$ extends from the end of the arm vector $\vec{p_a}$ to the specified gripper position in the direction specified by the approach vector. It is possible that the approach vector indicates an alignment of the wrist whose starting end cannot be attained by the arm vector. Tests must be derived here to address this concern.

The magnitude of the arm in Figure 5.5 can be seen to be a function of only the arm offsets a_2 and a_3 and control variable θ_3. Note that while control variables θ_1 and θ_2 influence the direction of the arm vector $\vec{p_a}$, they do not affect its magnitude. Since a_2 and a_3 are constants, the magnitude of $\vec{p_a}$ changes when and only when θ_3 changes. The magnitude of vector $\vec{p_a}$ is of course

$$|\vec{p_a}| = \sqrt{{p_{a_x}}^2 + {p_{a_y}}^2 + {p_{a_z}}^2} \tag{5.44}$$

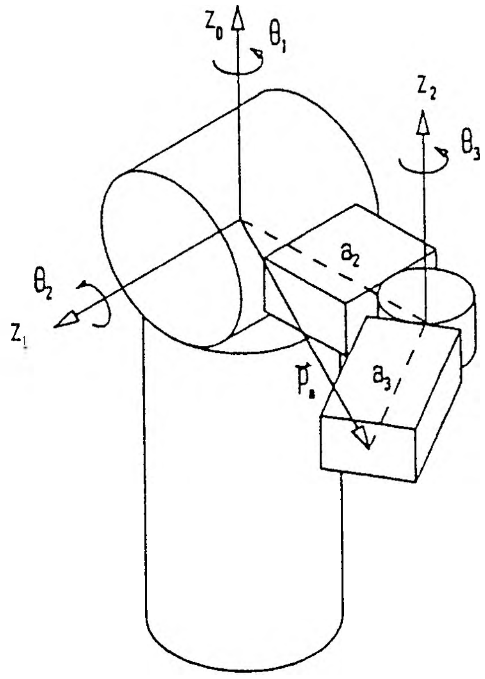


Figure 5.5. Envelope of the Arm Proper

It is clear from the figure that $\vec{p_a}$ will take on its greatest magnitude when θ_3 is zero and the arm is fully extended.

$$|\vec{p_a}|_{\max} = a_2 + a_3 \tag{5.45}$$

As θ_3 is capable of 180° of movement, from −90° to +90°, the magnitude of $\vec{p_a}$ will be smallest when θ_3 takes on either −90° or +90° as its value; vector $\vec{p_a}$ will form the hypotenuse of a right triangle for which a_2 and a_3 are the lengths of the sides.

$$|\vec{p_a}|_{\min} = \sqrt{{a_2}^2 + {a_3}^2} \tag{5.46}$$

The calculation for the arm offsets of 100 mm each yield a minimum magnitude of 141.421 mm.

As with the position vector components, the constraints upon the joint will be ignored here to allow for the solution of a wider range of position and orientation combinations; violations of joint constraints will be detected later. If joint 3 is allowed full freedom of movement, the magnitude of $\vec{p_a}$ can be seen to become zero when the arm folds back upon itself. Unfortunately, as will be seen later, if the magnitude of $\vec{p_a}$ is allowed to take on the value zero itself, the solution process will break down; this particular circumstance must then be avoided. Another breakdown occurs if only the x and y components of $\vec{p_a}$ become zero. Due to the configuration of the arm, this would only occur in the previous case or when $\vec{p_a}$ is directed vertically up or down; component p_z would become $\pm(a_2 + a_3 + d_5)$. This second situation must also be avoided. The two situations share the condition of both p_x and p_y being zero and thus may both be detected with a single comparison. As neither of these peculiar arm orientations is achievable anyway, they may both be detected here, along with the exceedance of the vector magnitude discussed previously.

If any of these conditions is not met, no further work should be done towards a solution as none can exist. If on the other hand the desired magnitude of $\vec{p_a}$ does meet these specifications, there will be solution sets for θ_1, θ_2, and θ_3 which would position the arm vector $\vec{p_a}$ as desired; again, however, they may not be within the ranges imposed by the physical parameters of the Armatron manipulator discussed in the previous chapter.

2. Numerical Example. Continuing the numerical example given by Equation (5.15), the components of the wrist vector are found using Equations (5.38), (5.39), and (5.40), and the components of vector $\bar{a}$. It should be noted that here and in the subsequent sections of this chapter the arithmetic will be carried out to three decimal places; this will lead to slight deviations with the results obtained by computer.

$$p_{w_x} = d_5 a_x \tag{5.47}$$

$$p_{w_x} = 100(0.331) = 33.100 \tag{5.48}$$

$$p_{w_y} = d_5 a_y \tag{5.49}$$

$$p_{w_y} = 100(-0.934) = -93.400 \tag{5.50}$$

$$p_{w_z} = d_5 a_z \tag{5.51}$$

$$p_{w_z} = 100(-0.137) = -13.700 \tag{5.52}$$

The arm vector components are then found using Equations (5.41), (5.42), and (5.43), along with the components of the vector $\bar{p}$ from Equation (5.15) and those just derived for vector $\bar{p}_w$.

$$p_{a_x} = p_x - p_{w_x} \tag{5.53}$$

$$p_{a_x} = 39.566 - 33.100 = 6.466 \tag{5.54}$$

$$p_{a_y} = p_y - p_{w_y} \tag{5.55}$$

$$p_{a_y} = -260.692 - (-93.400) = -167.292 \tag{5.56}$$

$$p_{a_z} = p_z - p_{w_z} \tag{5.57}$$

$$p_{a_z} = 55.745 - (-13.700) = 69.445 \tag{5.58}$$

The maximum obtainable magnitude of $\vec{p_a}$ was given by Equation (5.45).

$$|\vec{p_a}|_{\max} = a_2 + a_3 \tag{5.59}$$

$$|\vec{p_a}|_{\max} = 100 + 100 = 200 \tag{5.60}$$

The magnitude of the desired position vector is given by Equation (5.44) and the components of Equations (5.54), (5.56), and (5.58).

$$|\vec{p_a}| = \sqrt{{p_{a_x}}^2 + {p_{a_y}}^2 + {p_{a_z}}^2} \tag{5.61}$$

$$|\vec{p_a}| = \sqrt{(6.466)^2 + (-167.292)^2 + (69.445)^2} = 181.249 \tag{5.62}$$

This value is indeed less than the maximum possible for the arm. Further, the components p_{ax} and p_{ay} are not both simultaneously zero, so numerical solutions will exist for the arm control variables.

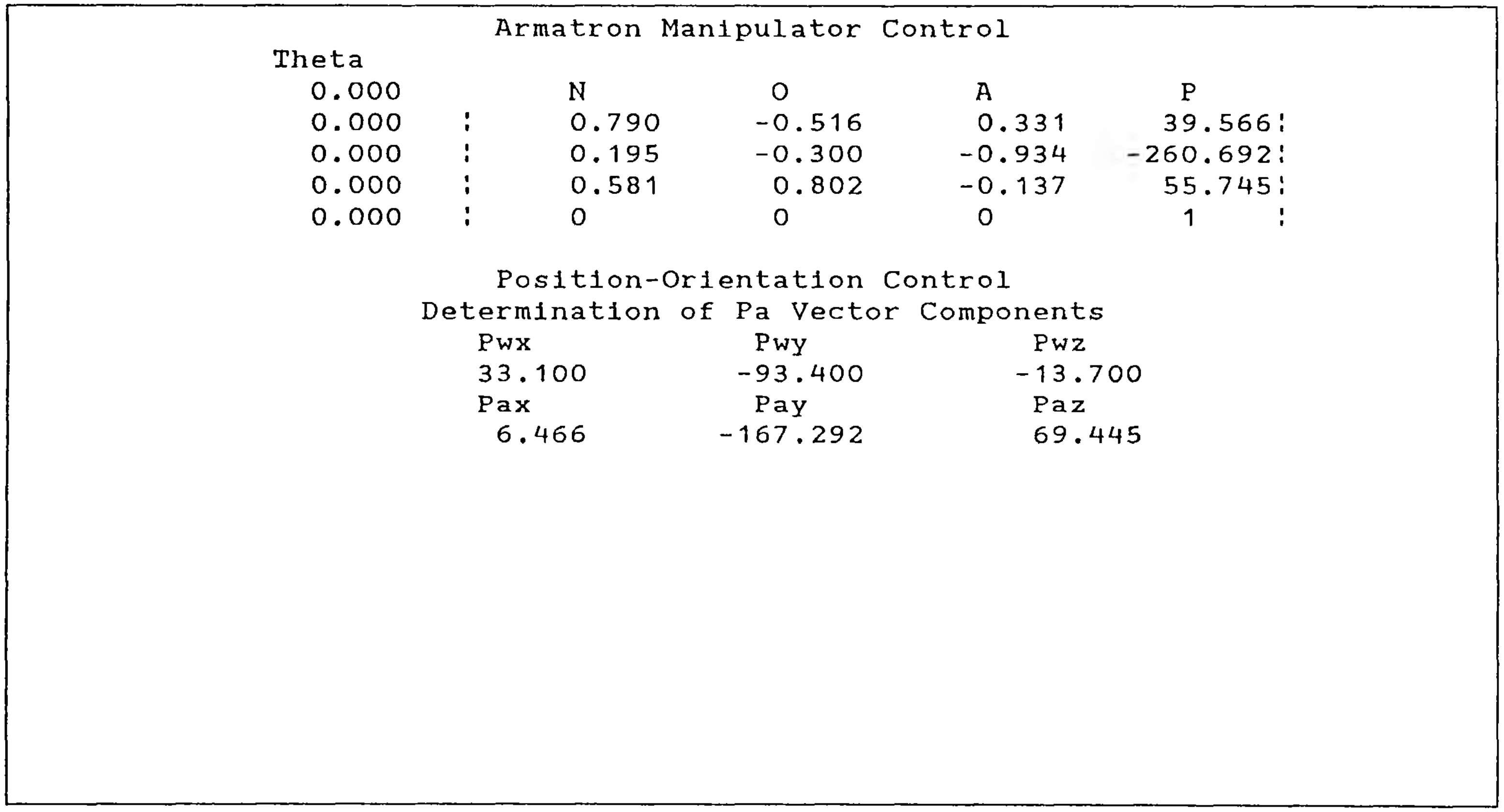
3. Program Structure. Procedure `calc_arm_end` determines the components of the arm vector in a straightforward fashion. The wrist vector components are calculated using Equations (5.38), (5.39), and (5.40) within a loop, as are Equations (5.41), (5.42), and (5.43) for calculation of the arm vector components.

```
for (i = 0; i <= 2; i++)
   {
   pw[i] = d5 * noap[2][i];
   lcprintf (arm_row, arm_cols[i], pw[i]);
   }
for (i = 0; i <= 2; i++)
```

```
   {
   pa[i] = noap[3][i] - pw[i];
   lcprintf (arm_row+2, arm_cols[i], pa[i]);
   }
mag_pa = magnitude (pa);
if ( (mag_pa > a2+a3) ¦ ( (fabs(pa[0]) < tolerance) &
                          (fabs(pa[1]) < tolerance) ) )
      mag_ok = 0;
   else
      mag_ok = 1;
return (mag_ok);
```

This procedure concludes by examining the generated components for the arm vector to insure that they meet the specifications described in the derivation of equations. A value is returned to the invoking procedure indicating the status of this examination.

4. Program Example. Execution of procedure `calc_arm_end` leads to the display of the information shown in Figure 5.6.



```
                    Armatron Manipulator Control
          Theta
           0.000            N          O          A          P
           0.000    :     0.790     -0.516      0.331     39.566:
           0.000    :     0.195     -0.300     -0.934   -260.692:
           0.000    :     0.581      0.802     -0.137     55.745:
           0.000    :     0               0          0          1     :

                    Position-Orientation Control
                Determination of Pa Vector Components
                   Pwx              Pwy              Pwz
                   33.100         -93.400          -13.700
                   Pax              Pay              Paz
                    6.466        -167.292           69.445
```

Figure 5.6. Display for Wrist and Arm Vector Components

D. ARM CONTROL VARIABLES

1. Derivation of Equations. Recall from the previous chapter the derivation of the link transformation A matrices. As explained there, link matrix i may be interpreted as transforming link i and those following it with respect to link i's coordinate frame. The matrix product $A_1A_2A_3$ results in a transformation which will translate and rotate the base coordinate frame to that of the end of the third link. Multiplication of this transform by the normal, orientation, and approach vectors as well as the base origin was shown to result in the transformation matrix itself, as the vectors and point are represented by the identity matrix. Thus, the following relationship holds for the components of the orientation vectors $\vec{n}_a$, $\vec{o}_a$, and $\vec{a}_a$, and translation point p_a at the end of the third link, or the arm proper:

$$\begin{bmatrix} n_{a_x} & o_{a_x} & a_{a_x} & p_{a_x} \\ n_{a_y} & o_{a_y} & a_{a_y} & p_{a_y} \\ n_{a_z} & o_{a_z} & a_{a_z} & p_{a_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_1A_2A_3 \tag{5.63}$$

The product of the matrix multiplication $A_1A_2A_3$ is obtained from Equation (4.72) in the previous chapter and substituted into Equation (5.63).

$$\begin{bmatrix} n_{a_x} & o_{a_x} & a_{a_x} & p_{a_x} \\ n_{a_y} & o_{a_y} & a_{a_y} & p_{a_y} \\ n_{a_z} & o_{a_z} & a_{a_z} & p_{a_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_1s_2 & c_1c_2s_3 + s_1c_3 & a_3(c_1c_2c_3 - s_1s_3) + a_2c_1c_2 \\ s_1c_2c_3 + c_1s_3 & -s_1s_2 & s_1c_2s_3 - c_1c_3 & a_3(s_1c_2c_3 + c_1s_3) + a_2s_1c_2 \\ s_2c_3 & c_2 & s_2s_3 & (a_3c_3 + a_2)s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.64}$$

At this point, nothing is known about the components of vectors $\vec{n}_a$, $\vec{o}_a$, or $\vec{a}_a$ or any of the three control variables; however, the previous section established values for p_{a_x}, p_{a_y}, and p_{a_z}. Thus three equations may be obtained from Equation (5.64) by equating elements (1,4), (2,4), and (3,4).

$$p_{a_x} = a_3(c_1c_2c_3 - s_1s_3) + a_2c_1c_2 \tag{5.65}$$

$$p_{a_y} = a_3(s_1c_2c_3 + c_1s_3) + a_2s_1c_2 \tag{5.66}$$

$$p_{a_z} = (a_3c_3 + a_2)s_2 \tag{5.67}$$

The task is now one of solving three equations of three unknowns; the complicating factor is of course that the three unknowns are present as arguments of the sine and cosine functions. One technique to begin the solution process is to square both sides of equations, as there are numerous sines and cosines involved and some will inevitably combine when squared and added to form one by trigonometric substitution. Equations (5.65) and (5.66) are squared first; the resulting equations are observed to have complementary terms and are thus added.

$$p_{a_x}{}^2 = c_1^2 c_2^2 (a_3 c_3 + a_2)^2 - 2a_3 c_1 s_1 c_2 s_3 (a_3 c_3 + a_2) + a_3^2 s_1^2 s_3^2 \tag{5.68}$$

$$p_{a_y}{}^2 = s_1^2 c_2^2 (a_3 c_3 + a_2)^2 + 2a_3 c_1 s_1 c_2 s_3 (a_3 c_3 + a_2) + a_3^2 c_1^2 s_3^2 \tag{5.69}$$

$$\begin{aligned} p_{a_x}{}^2 + p_{a_y}{}^2 = {} & c_1^2 c_2^2 (a_3 c_3 + a_2)^2 + s_1^2 c_2^2 (a_3 c_3 + a_2)^2 \\ & - 2a_3 c_1 s_1 c_2 s_3 (a_3 c_3 + a_2) + 2a_3 c_1 s_1 c_2 s_3 (a_3 c_3 + a_2) + a_3^2 s_1^2 s_3^2 + a_3^2 c_1^2 s_3^2 \end{aligned} \tag{5.70}$$

$$p_{a_x}{}^2 + p_{a_y}{}^2 = c_1^2 c_2^2 (a_3 c_3 + a_2)^2 + s_1^2 c_2^2 (a_3 c_3 + a_2)^2 + a_3^2 s_1^2 s_3^2 + a_3^2 c_1^2 s_3^2 \tag{5.71}$$

Equation (5.71) then has the common factors of the first and second pairs of right hand side terms factored; trigonometric substitution then reduces the equation.

$$p_{a_x}{}^2 + p_{a_y}{}^2 = c_2^2 (a_3 c_3 + a_2)^2 (c_1^2 + s_1^2) + a_3^2 s_3^2 (s_1^2 + c_1^2) \tag{5.72}$$

$$p_{a_x}{}^2 + p_{a_y}{}^2 = c_2^2 (a_3 c_3 + a_2)^2 + a_3^2 s_3^2 \tag{5.73}$$

The expression of Equation (5.67) when squared will combine with the first term of Equation (5.73)'s right hand side by trigonometric substitution upon addition.

$$p_{a_z}{}^2 = s_2^2 (a_3 c_3 + a_2)^2 \tag{5.74}$$

$$p_{a_x}{}^2 + p_{a_y}{}^2 + p_{a_z}{}^2 = c_2^2 (a_3 c_3 + a_2)^2 + s_2^2 (a_3 c_3 + a_2)^2 + a_3^2 s_3^2 \tag{5.75}$$

$$p_{a_x}{}^2 + p_{a_y}{}^2 + p_{a_z}{}^2 = (a_3 c_3 + a_2)^2 (c_2^2 + s_2^2) + a_3^2 s_3^2 \tag{5.76}$$

$$p_{a_x}{}^2 + p_{a_y}{}^2 + p_{a_z}{}^2 = (a_3 c_3 + a_2)^2 + a_3^2 s_3^2 \tag{5.77}$$

The right hand side is then fully expanded and factored again.

$$p_{a_x}{}^2 + p_{a_y}{}^2 + p_{a_z}{}^2 = a_3^2 c_3^2 + 2a_2 a_3 c_3 + a_2^2 + a_3^2 s_3^2 \tag{5.78}$$

$$p_{a_x}{}^2 + p_{a_y}{}^2 + p_{a_z}{}^2 = a_3^2(c_3^2 + s_3^2) + 2a_2a_3c_3 + a_2^2 \tag{5.79}$$

Trigonometric substitution is applied once more and the equation is solved for the cosine of θ_3.

$$p_{a_x}{}^2 + p_{a_y}{}^2 + p_{a_z}{}^2 = a_3^2 + 2a_2a_3c_3 + a_2^2 \tag{5.80}$$

$$c_3 = \frac{p_{a_x}{}^2 + p_{a_y}{}^2 + p_{a_z}{}^2 - a_3^2 - a_2^2}{2a_2a_3} \tag{5.81}$$

The right hand side of Equation (5.81) is an expression of known quantities. The magnitude of the sine of the angle is then known, which leads to an expression for the angle itself.

$$\sin\theta_3 = \pm\sqrt{1 - \cos^2\theta_3} \tag{5.82}$$

$$\theta_3 = \tan^{-1}\left(\frac{\sin\theta_3}{\cos\theta_3}\right) \tag{5.83}$$

The cosine of the angle, given by Equation (5.81), will indicate by its sign whether the angle itself is in the first and fourth quadrants (positive) or in the second and third quadrants (negative). Note however that the sign of the angle's sine is not specified by Equation (5.82). For a given cosine, this situation yields two possible sine values, as depicted in Figure 5.7. The cosine of angle γ is the x-coordinate of the two points (x_1, y_1) and (x_2, y_2) on the unit circle about the origin; this is true because cos(γ) = cos(-γ). The y-coordinates y_1 and y_2 correspond to the two results generated for the sine of the angle by Equation (5.82).

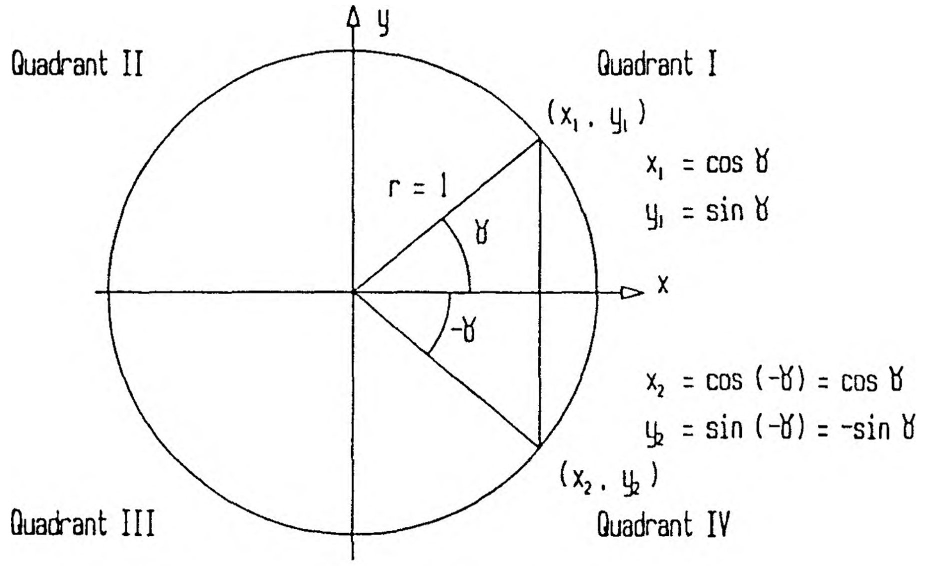


Figure 5.7. Possible Sines for a Given Cosine

Now consider the result of the arc tangent function used in Equation (5.83) and throughout this chapter. The typical arc tangent function returns a first quadrant angle, 0^R to $\frac{\pi^R}{2}$, if its argument is positive; a fourth quadrant angle, $-\frac{\pi^R}{2}$ to 0^R, is returned for a negative argument. The cosine of an angle in the first quadrant is positive, as is its sine; the tangent of this angle is thus also positive so the arc tangent function is justified in returning a first quadrant angle for a positive argument. The cosine of an angle in the fourth quadrant is also positive, but its sine is negative; thus the tangent of the angle is negative, so the arc tangent function is also justified in returning a fourth quadrant angle for a negative tangent. In both of the preceding cases, the cosine was positive; consider now a negative cosine. A positive sine creates a negative tangent, for which the arc tangent returns an angle in the fourth quadrant, instead of the desired second quadrant angle. Likewise, a negative sine produces a positive tangent, and a first quadrant angle is returned by the arc tangent function instead of the correct third quadrant angle. Consider the cosine and sine of a first quadrant angle γ as the coordinates (x_1, y_1) of a point on a unit circle centered about the origin; see Figure 5.8. It is clear that along the line through the first point and the origin there is a point in the third quadrant whose coordinates (x_2, y_2) are equal in magnitude but opposite in sign to those of the first point; thus there is an angle in the third quadrant π^R away from the first angle whose tangent is equal to that of the first angle. This possibility must also be considered. The same argument holds true for an angle in the fourth quadrant; its cosine is positive, its sine is negative, and its tangent is thus negative. There is another angle π^R away in the second quadrant whose cosine and sine are equal but reversed in sign. Thus, if the cosine of Equation (5.81) is negative, both of the results of the arc tangent must be adjusted.

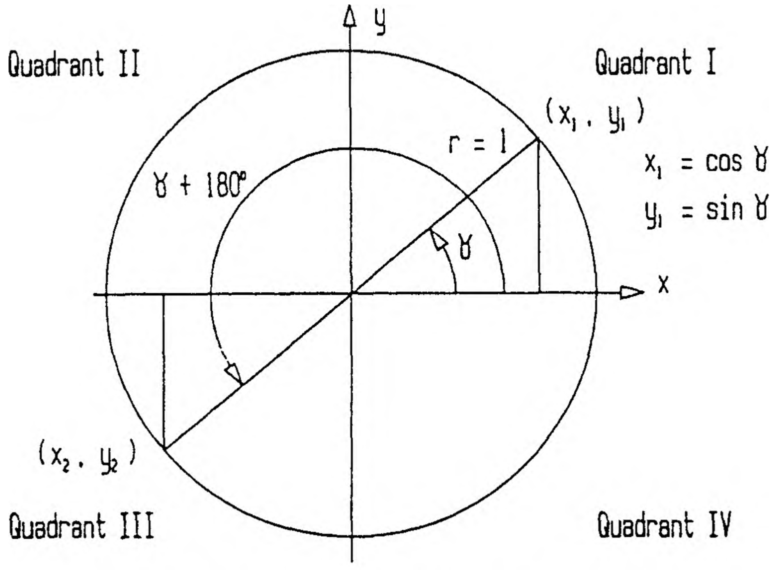


Figure 5.8. Geometry of Equal Tangents

The adjustment for the arc tangent result when a negative cosine is involved could be as simple as the addition of π^R, as described above. When a fourth quadrant angle is returned by the arc tangent function, the addition of π^R will yield a second quadrant angle, $\frac{\pi^R}{2}$ to π^R, or 90° to 180°. In the case of a first quadrant angle being returned, the addition of π^R will yield a third quadrant angle, π^R to $\frac{3\pi^R}{2}$, or 180° to 270°. However, since the constraints on a control variable are typically stated as a negative value running through to a positive value, it will facilitate comparisons with bounds later if the control variables are in these ranges. This point should be considered for any joint in any robot arm, not just the one under consideration. The corrective measure for a third quadrant angle whose cosine and sine are both negative will be to subtract π^R from the first quadrant arc tangent result, rather than adding it, so that a value in the range $-\pi^R$ to $\frac{-\pi^R}{2}$ is obtained. The adjustment for a second quadrant angle remains the addition of π^R to the fourth quadrant arc tangent result.

Lastly, note that if the cosine of θ_3 is zero, then the angle is either $\frac{\pi^R}{2}$ or $\frac{-\pi^R}{2}$. As the sine of the angle is not known, either of these is possible; they simply replace the two solutions otherwise obtained by Equation (5.83).

Consider again the expressions for p_{ax} and p_{ay} from Equations (5.65) and (5.66), respectively, in the search for a solution for one of the remaining angles.

$$p_{a_x} = c_1c_2(a_3c_3 + a_2) - a_3s_1s_3 \tag{5.84}$$

$$p_{a_y} = s_1c_2(a_3c_3 + a_2) + a_3c_1s_3 \tag{5.85}$$

The first term of the right hand side of each can be eliminated by multiplying the first by $-s_1$ and the second by c_1 and then adding.

$$-p_{a_x}s_1 = -\,c_1s_1c_2(a_3c_3 + a_2) + a_3s_1^2s_3 \tag{5.86}$$

$$p_{a_y}c_1 = c_1s_1c_2(a_3c_3 + a_2) + a_3c_1^2s_3 \tag{5.87}$$

$$-p_{a_x}s_1 + p_{a_y}c_1 = a_3s_1^2s_3 + a_3c_1^2s_3 \tag{5.88}$$

Factoring the right hand side and again using trigonometric substitution,

$$-p_{a_x}s_1 + p_{a_y}c_1 = a_3s_3(s_1^2 + c_1^2) \tag{5.89}$$

$$-p_{a_x}s_1 + p_{a_y}c_1 = a_3s_3 \tag{5.90}$$

Since θ_3 has been determined previously, this equation has θ_1 as its single unknown variable; however, the equation involves both the cosine and sine of the angle. A technique for solving an equation of this type is to view the coefficients of the cosine and sine terms as the lengths of the legs of a right triangle. Consider the relationships below as pictured in Figure 5.9.

$$r = +\sqrt{{p_{a_x}}^2 + {p_{a_y}}^2} \tag{5.91}$$

$$p_{a_x} = r \cos \beta \tag{5.92}$$

$$p_{a_y} = r \sin \beta \tag{5.93}$$

$$\beta = \tan^{-1}\left(\frac{p_{a_y}}{p_{a_x}} \right) \tag{5.94}$$

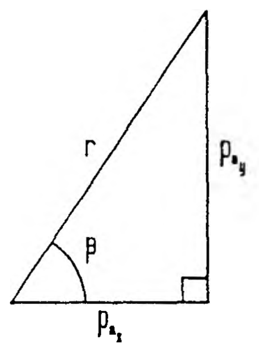


Figure 5.9. Geometric Configuration for Solution of Equation Containing Sine and Cosine Terms

Note that the angle β in Figure 5.9 was deliberately chosen as that opposite the side representing the coefficient of the cosine in Equation (5.90).

The expressions for p_{a_x} and p_{a_y} from Equations (5.92) and (5.93), respectively, are substituted into Equation (5.90) to form the following:

$$-(r\cos\beta)\sin\theta_1 + (r\sin\beta)\cos\theta_1 = a_3 s_3 \tag{5.95}$$

This equation is then factored and a trigonometric substitution applied to obtain a relationship involving θ_1.

$$r(\sin\beta\cos\theta_1 - \cos\beta\sin\theta_1) = a_3 s_3 \tag{5.96}$$

$$r\sin(\beta - \theta_1) = a_3 s_3 \tag{5.97}$$

$$\sin(\beta - \theta_1) = \frac{a_3 s_3}{+\sqrt{{p_{a_x}}^2 + {p_{a_y}}^2}} \tag{5.98}$$

The variable r has been replaced by use of Equation (5.91). The cosine of the angular difference is of course

$$\cos(\beta - \theta_1) = \pm\sqrt{1 - \sin^2(\beta - \theta_1)} \tag{5.99}$$

As Equation (5.98) involves a division, the possibility of a division by zero must be considered. The divisor in the division is $\sqrt{{p_{a_x}}^2 + {p_{a_y}}^2}$, which approaches zero only when both p_{a_x} and p_{a_y} become close to zero. This exceptional situation is one of two pointed out upon determination of the arm end point coordinates. As noted there, even ignoring joint variable restraints, the arm configuration prevents this situation from occurring unless the arm folds back on itself at the elbow or is fully extended vertically up or down. Each of these arm configurations (which are unattainable due to joint restrictions at any rate) are detected at that point; the solution process does not continue past that point. Thus no special measures need to be taken here to accommodate this possibility.

Now consider the sine of the angular difference $\beta - \theta_1$ given by Equation (5.98). The sign of this quantity is strictly dependent upon the sine of θ_3 since the remainder of the factors, a_3 and $\frac{1}{\sqrt{p_{o_x}^2 + p_{o_y}^2}}$, are positive. When θ_3 is between 0^R and π^R, its sine is positive and the sine of $\beta - \theta_1$ must likewise be positive. This in turn implies that $\beta - \theta_1$ is between 0^R and π^R. By the same reasoning, when θ_3 is between $-\pi^R$ and 0^R, the sine of $\beta - \theta_1$ is similarly negative, forcing $\beta - \theta_1$ to be between $-\pi^R$ and 0^R.

The result of Equation (5.98) is divided by that of (5.99) to obtain a relationship for the quantity $\beta - \theta_1$.

$$\frac{\sin(\beta - \theta_1)}{\cos(\beta - \theta_1)} = \tan(\beta - \theta_1) \tag{5.100}$$

$$\beta - \theta_1 = \tan^{-1}\left(\frac{\sin(\beta - \theta_1)}{\cos(\beta - \theta_1)} \right) \tag{5.101}$$

Now consider the necessary adjustments to the arc tangent results. As described previously, if the sine of θ_3 is a positive quantity, then so is the sine of $\beta - \theta_1$, and the difference $\beta - \theta_1$ must lie in the first or second quadrant. The sign of the difference's cosine cannot be determined, so two possible values for $\beta - \theta_1$ must be considered as there are two angles with the same sine and magnitude of cosine. The same argument holds for a negative sine of θ_3. The arc tangent function of Equation (5.101) will be employed for two arguments, returning a first quadrant angle for the positive argument and a fourth quadrant angle for the negative argument. Therefore, if the sine of θ_3 is positive, the addition of π^R to the fourth quadrant angle will produce the desired second quadrant angle; see Figure 5.10.

A similar adjustment is made if the sine of θ_3 is negative. The addition of π^R to the first quadrant angle will yield the desired fourth quadrant angle; see Figure 5.11.

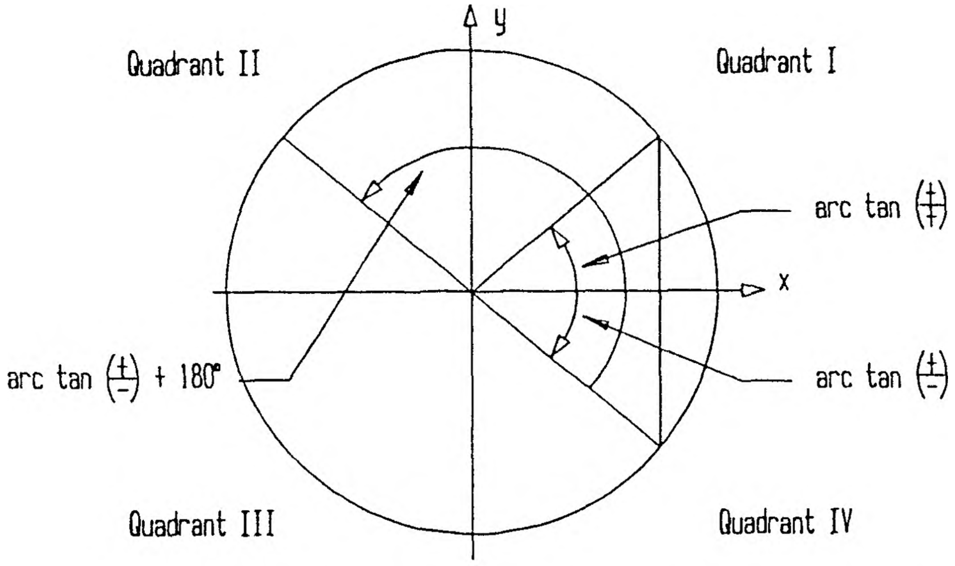


Figure 5.10. Adjusted Arc Tangent Results for a Positive Sine

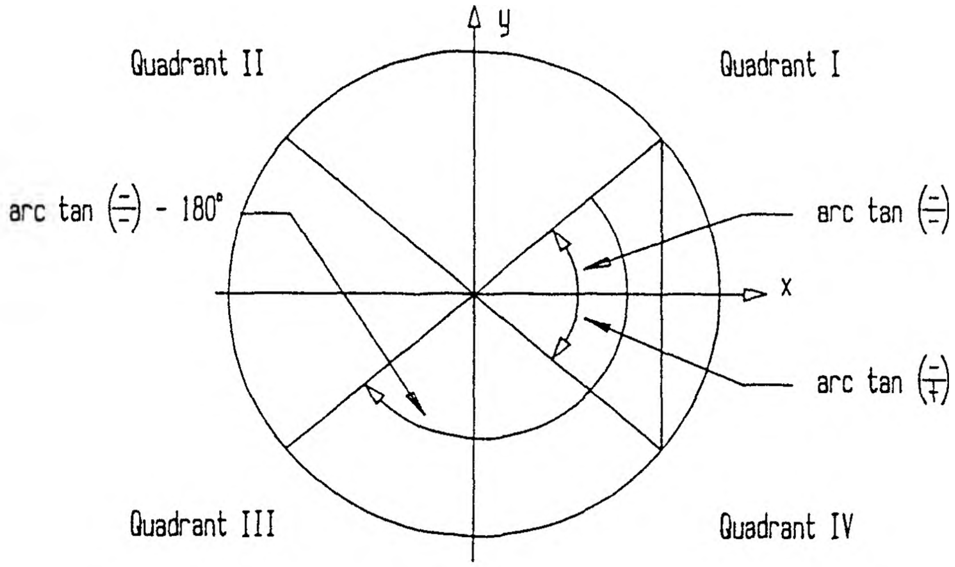


Figure 5.11. Adjusted Arc Tangent Results for a Negative Sine

The control variable θ_1 itself will be adjusted to the range conventions after its calculation from the quantities $\beta - \theta_1$ and β. Note also that if the cosine of Equation (5.99) is zero, the angle is uniquely identified by the sine as $\frac{\pm\,\pi^R}{2}$.

Now consider the angle β. Its cosine and sine are derived from Equations (5.92), (5.93), and (5.91). The relationship for the angle itself was given by Equation (5.94).

$$\cos\beta = \frac{p_{a_x}}{+\sqrt{{p_{a_x}}^2 + {p_{a_y}}^2}} \tag{5.102}$$

$$\sin\beta = \frac{p_{a_y}}{+\sqrt{{p_{a_x}}^2 + {p_{a_y}}^2}} \tag{5.103}$$

$$\beta = \tan^{-1}\left(\frac{p_{a_y}}{p_{a_x}}\right) \tag{5.104}$$

Since the signs of the cosine and sine are known, there is only one possible solution for β. As with $\beta - \theta_1$, the arc tangent result is adjusted when the cosine of β is negative by the addition of π^R. The difference here is that there is only a single resulting angle for β, as one value each is given for the cosine and sine. Also, Equation (5.102) shows that the cosine of β will only be zero if p_{a_x} is zero. In this case, the sign of the sine, or p_{a_y}, will indicate either $\frac{\pm\pi^R}{2}$ as the angle.

With the angular difference $\beta - \theta_1$ and β alone determined, θ_1 follows.

$$\theta_1 = \beta - (\beta - \theta_1) \tag{5.105}$$

Each of the two θ_3 values results in two values for the quantity $\beta - \theta_1$ and consequently two values for θ_1 itself. As with θ_3, adjustments may be necessary for θ_1 to keep its values within the range -180° to $+180^\circ$. The values of θ_1 resultant from Equation (5.105) may be adjusted by simple comparisons with the boundary limits; one full circle, or 360°, should be subtracted for angles larger than 180°, and 360° is to be added for angles smaller than -180°.

The expressions for p_{a_x} and p_{a_y} of Equations (5.65) and (5.66), respectively, are returned to a second time to derive a relationship for θ_2.

$$p_{a_x} = c_1 c_2(a_3 c_3 + a_2) - a_3 s_1 s_3 \tag{5.106}$$

$$p_{a_y} = s_1 c_2(a_3 c_3 + a_2) + a_3 c_1 s_3 \tag{5.107}$$

In deriving an equation for θ_1, these equations were multiplied by $-s_1$ and c_1, respectively, to eliminate the first term of the right hand sides. Now consider multiplying them by c_1 and s_1, respectively. This will lead to the elimination of the second term of the right hand sides when the resulting equations are added.

$$p_{a_x} c_1 = c_1^2 c_2(a_3 c_3 + a_2) - a_3 c_1 s_1 s_3 \tag{5.108}$$

$$p_{a_y} s_1 = s_1^2 c_2(a_3 c_3 + a_2) + a_3 c_1 s_1 s_3 \tag{5.109}$$

$$p_{a_x} c_1 + p_{a_y} s_1 = c_1^2 c_2(a_3 c_3 + a_2) + s_1^2 c_2(a_3 c_3 + a_2) \tag{5.110}$$

Factoring the right hand side and applying trigonometric substitution leads to an equation for the cosine of θ_2, the only remaining unknown of the original three.

$$p_{a_x} c_1 + p_{a_y} s_1 = c_2(a_3 c_3 + a_2)(c_1^2 + s_1^2) \tag{5.111}$$

$$p_{a_x} c_1 + p_{a_y} s_1 = c_2(a_3 c_3 + a_2) \tag{5.112}$$

$$c_2 = \frac{p_{a_x} c_1 + p_{a_y} s_1}{a_3 c_3 + a_2} \tag{5.113}$$

The third original relationship, Equation (5.67), provides a solution for the sine of the angle θ_2.

$$p_{a_z} = s_2(a_3c_3 + a_2) \tag{5.114}$$

$$s_2 = \frac{p_{a_z}}{a_3c_3 + a_2} \tag{5.115}$$

The relationships for the cosine and sine of the second control variable given by Equations (5.113) and (5.115), respectively, present again a concern that has occurred before, specifically division by zero. The arm offset constants a_2 and a_3 were observed earlier to each be 100 mm; consequently, the expression $a_3c_3 + a_2$ can take on the value 0 if $c_3 = -1$. This is physically impossible, as θ_3 is the control variable for the elbow joint and is capable of only 90° of movement in either direction and cannot attain the required 180°. Consider the circumstances under which Equation (5.81), which defines c_3, would produce a value of -1.

$$c_3 = \frac{{p_{a_x}}^2 + {p_{a_y}}^2 + {p_{a_z}}^2 - a_3^2 - a_2^2}{2a_2a_3} \tag{5.116}$$

$$\frac{{p_{a_x}}^2 + {p_{a_y}}^2 + {p_{a_z}}^2 - a_3^2 - a_2^2}{2a_2a_3} = -1 \tag{5.117}$$

$${p_{a_x}}^2 + {p_{a_y}}^2 + {p_{a_z}}^2 - a_3^2 - a_2^2 = -2a_2a_3 \tag{5.118}$$

$${p_{a_x}}^2 + {p_{a_y}}^2 + {p_{a_z}}^2 = a_3^2 - 2a_2a_3 + a_2^2 \tag{5.119}$$

$${p_{a_x}}^2 + {p_{a_y}}^2 + {p_{a_z}}^2 = (a_3 - a_2)^2 \tag{5.120}$$

$${p_{a_x}}^2 + {p_{a_y}}^2 + {p_{a_z}}^2 = 0 \tag{5.121}$$

Clearly, the position of the end of the third link, or the start of the wrist, must coincide with the origin. This is the second of the two exceptional situations singled out upon determination of the end point of the three arm links. The tests performed during the solution for the arm vector components in the preceding section handle this situation so no additional tests need be performed here.

The result of Equation (5.115) is divided by the result of (5.113) to obtain a relationship for θ_2.

$$\frac{\sin \theta_2}{\cos \theta_2} = \tan \theta_2 \tag{5.122}$$

$$\theta_2 = \tan^{-1}\left(\frac{\sin \theta_2}{\cos \theta_2} \right) \tag{5.123}$$

As mentioned in previous solutions, the arc tangent used in Equation (5.123) returns an angle in the first or fourth quadrant. The results of Equations (5.113) and (5.115) are used for the determination of the proper quadrant of the single resultant θ_2. If the cosine is positive, then arc tangent returns the proper angle. If however the cosine is negative, then π^R must be added to the arc tangent result if the sine is positive or subtracted if the sine is negative. If the cosine of θ_2 in Equation (5.113) is zero, the sign of the sine in Equation (5.115) determines either $\frac{\pm \pi^R}{2}$ as the angle. Note that the evaluation of θ_2 requires both of the angles θ_1 and θ_3. The former had two independent solutions, and the latter had two solutions for each of these; thus there are four separate solutions for θ_2.

2. <u>Numerical Example</u>. The numerical example continues by calculating the first derived arm control variable, θ_3. Equation (5.81) specifies the cosine of the angle, and Equations (5.54), (5.56), and (5.58) provide the necessary values for the calculation.

$$\cos \theta_3 = \frac{{p_{a_x}}^2 + {p_{a_y}}^2 + {p_{a_z}}^2 - a_3^2 - a_2^2}{2a_2a_3} \tag{5.124}$$

$$\cos\theta_3 = \frac{(6.466)^2 + (-167.292)^2 + (69.445)^2 - 100^2 - 100^2}{2(100)(100)} = 0.643 \qquad (5.125)$$

The sine of the angle is then obtained from Equation (5.82).

$$\sin\theta_3 = \pm\sqrt{1 - \cos^2\theta_3} \qquad (5.126)$$

$$\sin\theta_3 = \pm\sqrt{1 - 0.643^2} = \pm 0.766 \qquad (5.127)$$

Equation (5.83) then yields the two possible values for θ_3.

$$\theta_3 = \tan^{-1}\left(\frac{\sin\theta_3}{\cos\theta_3}\right) \qquad (5.128)$$

$$\theta_3 = \tan^{-1}\left(\frac{\pm 0.766}{0.643}\right) = \tan^{-1}(\pm 1.191) \qquad (5.129)$$

$$\theta_3 = 49.989^\circ \text{ or } -49.989^\circ \qquad (5.130)$$

Since the cosine of the angle is positive, no adjustments need be made to these values; solutions for θ_3 must lie in the first or fourth quadrants.

Moving on to θ_1, the angular difference $\beta - \theta_1$ is first determined. When $\theta_3 = +49.989^\circ$, Equation (5.98) produces

$$\sin(\beta - \theta_1) = \frac{a_3 s_3}{+\sqrt{p_{a_x}{}^2 + p_{a_y}{}^2}} \qquad (5.131)$$

$$\sin(\beta - \theta_1) = \frac{100\sin(+49.989^\circ)}{+\sqrt{(6.466)^2 + (-167.292)^2}} = 0.457 \qquad (5.132)$$

When $\theta_3 = -49.989^\circ$, the sine becomes

$$\sin(\beta - \theta_1) = \frac{100 \sin(-49.989^\circ)}{+\sqrt{(6.466)^2 + (-167.292)^2}} = -0.457 \tag{5.133}$$

For both angles, the cosine given by Equation (5.99) is the same.

$$\cos(\beta - \theta_1) = \pm\sqrt{1 - \sin^2(\beta - \theta_1)} \tag{5.134}$$

$$\cos(\beta - \theta_1) = \pm\sqrt{1 - (\pm 0.457)^2} = \pm 0.889 \tag{5.135}$$

For $\theta_3 = +49.989^\circ$ and $\sin(\beta - \theta_1) = 0.457$, the possibilities for the angular difference are given by Equation (5.101).

$$\beta - \theta_1 = \tan^{-1}\left(\frac{\sin(\beta - \theta_1)}{\cos(\beta - \theta_1)}\right) \tag{5.136}$$

$$\beta - \theta_1 = \tan^{-1}\left(\frac{0.457}{\pm 0.889}\right) \tag{5.137}$$

$$\beta - \theta_1 = 27.206^\circ \text{ or } -27.206^\circ \tag{5.138}$$

Since the sine of the difference is known to be positive, 0° and 180° bound the angular difference; thus the second result of the arc tangent is adjusted by the addition of 180° from the fourth quadrant (positive cosine, negative sine, negative tangent) to the second quadrant (negative cosine, positive sine, negative tangent).

$$\beta - \theta_1 = 27.206^\circ \text{ or } -27.206^\circ + 180^\circ \tag{5.139}$$

$$\beta - \theta_1 = 27.206^\circ \text{ or } 152.794^\circ \tag{5.140}$$

For $\theta_3 = -49.989^\circ$ and $\sin(\beta - \theta_1) = -0.457$

$$\beta - \theta_1 = \tan^{-1}\left(\frac{-0.457}{\pm 0.889} \right) \tag{5.141}$$

$$\beta - \theta_1 = -27.206^\circ \text{ or } 27.206^\circ \tag{5.142}$$

In this case, the sine of the difference is known to be negative, so the second result of the arc tangent is adjusted by the addition of 180° from the first quadrant (positive cosine, positive sine, positive tangent) to the third quadrant (negative cosine, negative sine, positive tangent) to create two angles on the range -180° to 0°.

$$\beta - \theta_1 = -27.206^\circ \text{ or } 27.206 + 180^\circ \tag{5.143}$$

$$\beta - \theta_1 = -27.206^\circ \text{ or } 207.206^\circ \tag{5.144}$$

Next, the β component as defined by Equation (5.94) is determined using the results of Equations (5.56) and (5.54).

$$\beta = \tan^{-1}\left(\frac{p_{a_y}}{p_{a_x}} \right) \tag{5.145}$$

$$\beta = \tan^{-1}\left(\frac{-167.292}{6.466} \right) = -87.787^\circ \tag{5.146}$$

Now consider the cosine and sine of β as defined by Equations (5.102) and (5.103).

$$\cos\beta = \frac{p_{a_x}}{+\sqrt{p_{a_x}^{\ 2} + p_{a_y}^{\ 2}}} \tag{5.147}$$

$$\sin\beta = \frac{p_{a_y}}{+\sqrt{p_{a_x}^{\ 2} + p_{a_y}^{\ 2}}} \tag{5.148}$$

Clearly, the signs of β's cosine and sine are identically those of p_{o_x} and p_{o_y}, respectively. Consequently, the cosine of β is positive and the sine is negative; the angle must then lie in the fourth quadrant. Since the cosine is positive, the value obtained above does not need to be adjusted by the addition of 180°.

The angle θ_1 is now determined using Equation (5.105).

$$\theta_1 = \beta - (\beta - \theta_1) \tag{5.149}$$

For $\theta_3 = 49.989°$, substitutions are made from Equations (5.146) and (5.140).

$$\theta_{1_1} = -87.787^\circ - (27.206^\circ) = -114.993^\circ \tag{5.150}$$

$$\theta_{1_2} = -87.787^\circ - (152.794^\circ) = -240.581^\circ \tag{5.151}$$

Note that the second value for θ_1 is outside of the desired limits $-180°$ and $+180°$, so an adjustment needs to be made. In this case, it is the addition of one full circle.

$$\theta_{1_2} = -240.581^\circ + 360^\circ = 119.419^\circ \tag{5.152}$$

For $\theta_3 = -49.989°$, substitutions are made from Equations (5.146) and (5.144).

$$\theta_{1_3} = -87.787^\circ - (-27.206^\circ) = -60.581^\circ \tag{5.153}$$

$$\theta_{1_4} = -87.787^\circ - (207.206^\circ) = -294.993^\circ \tag{5.154}$$

The second solution here is also outside the desired limits of $-180°$ and $+180°$, and since it is negative, it is adjusted by the addition of one full circle.

$$\theta_{1_4} = -294.993^\circ + 360^\circ = 65.007^\circ \tag{5.155}$$

Summarizing, the two θ_1 solutions for $\theta_3 = 49.989^\circ$ are -114.993° and 119.419°, and the two θ_1 solutions for $\theta_3 = -49.989^\circ$ are -60.581° and 65.007°.

Finally, θ_2 is evaluated using Equation (5.123).

$$\theta_2 = \tan^{-1}\left(\frac{s_2}{c_2}\right) \tag{5.156}$$

The cosine and sine of θ_2 are of course required prior to the use of Equation (5.123); they are also used to determine the proper quadrant of θ_2. From Equations (5.113) and (5.115),

$$c_2 = \frac{p_{a_x}c_1 + p_{a_y}s_1}{a_3c_3 + a_2} \tag{5.157}$$

$$s_2 = \frac{p_{a_z}}{a_3c_3 + a_2} \tag{5.158}$$

It was pointed out during the derivation of the equations that if θ_3 should become $\pm 180^\circ$, folding the arm back on itself at the elbow, its cosine would become -1 and the denominators of the formulas for θ_2's cosine and sine would become zero; this is not the case for this example.

Equations (5.54), (5.56), and (5.58) provide the coefficients p_{a_x}, p_{a_y}, and p_{a_z}, respectively. For $\theta_3 = 49.989^\circ$ and $\theta_1 = -114.993^\circ$,

$$c_2 = \frac{6.466\cos(-114.993^\circ) + (-167.292)\sin(-114.993^\circ)}{100\cos(49.989^\circ) + 100} = 0.906 \tag{5.159}$$

$$s_2 = \frac{69.445}{100\cos(49.989^\circ) + 100} = 0.423 \tag{5.160}$$

$$\theta_{2_1} = \tan^{-1}\left(\frac{0.423}{0.906}\right) = 25.027^\circ \qquad (5.161)$$

Since the cosine is positive, the desired result is a first or fourth quadrant angle and no adjustment need be made to the result of the arc tangent.

The example continues with $\theta_3 = 49.989^\circ$ and $\theta_1 = 119.419^\circ$. Note that since cosine is an even function and the two values for θ_3 differ only in sign, the value for the sine of θ_2 will not change throughout the example and need not be recalculated.

$$c_2 = \frac{6.466\cos(119.419^\circ) + (-167.292)\sin(119.419^\circ)}{100\cos(49.989^\circ) + 100} = -0.906 \qquad (5.162)$$

$$\theta_{2_2} = \tan^{-1}\left(\frac{0.423}{-0.906}\right) = -25.027^\circ \qquad (5.163)$$

Since the cosine of θ_2 is negative, the desired result is a second or third quadrant angle, and the result of the arc tangent function must be modified. Further, since the sine of the angle is positive, it is in the second quadrant, so 180° is added to the fourth quadrant arc tangent result to obtain the correct angle.

$$\theta_2 = -25.027^\circ + 180^\circ = 154.973^\circ \qquad (5.164)$$

For $\theta_3 = -49.989^\circ$ and $\theta_1 = -60.581^\circ$,

$$c_2 = \frac{6.466\cos(-60.581^\circ) + (-167.292)\sin(-60.581^\circ)}{100\cos(-49.989^\circ) + 100} = 0.906 \qquad (5.165)$$

$$\theta_{2_3} = \tan^{-1}\left(\frac{0.423}{0.906}\right) = 25.027^\circ \qquad (5.166)$$

Since the cosine of this θ_2 is positive, the result of the arc tangent does not need to be modified.

For $\theta_3 = -49.989^\circ$ and $\theta_1 = 65.007^\circ$,

$$c_2 = \frac{6.466\cos(65.007^\circ) + (-167.292)\sin(65.007^\circ)}{100\cos(-49.989^\circ) + 100} = -0.906 \tag{5.167}$$

$$\theta_{2_4} = \tan^{-1}\left(\frac{0.423}{-0.906}\right) = -25.027^\circ \tag{5.168}$$

Since the cosine of θ_2 is negative and the sine is positive, the angle is in the second quadrant; thus a correction factor of 180° is added to the arc tangent result to achieve the correct angle.

$$\theta_{2_4} = -25.027^\circ + 180^\circ = 154.973^\circ \tag{5.169}$$

This completes the possible solutions for θ_1, θ_2, and θ_3. The solution sets are summarized in Table 5.I.

Table 5.I. SOLUTION TRIPLES FOR ARM CONTROL VARIABLES

Control Variable	Set 1	Set 2	Set3	Set 4
1	-114.993	119.419	-60.581	65.007
2	25.027	154.973	25.027	154.973
3	49.989	49.989	-49.989	-49.989

Each of these control variable triples will achieve the desired position for the end point of the arm's third link. This is demonstrated geometrically on the coordinate frames of Figure 5.12 (a). The set of rotations specified by the first triple is depicted in Figure 5.12 (b). The first rotation is -114.993° about the z_0-axis, transforming links 1, 2, and 3. The next rotation is 25.027° about the z_1-axis, transforming only links 2 and 3. Finally, the z_2-axis is rotated about by 49.989°, and only link 3 moves here. In picturing these rotations, recall the right hand rule, which states that when the thumb of the right hand is laid parallel along the positive direction of the axis of rotation, the fingers curl about the axis in the direction of a positive rotation. The three remaining solutions are carried out in parts (c), (d), and (e) of Figure 5.12. Each solution can be seen to achieve the same position for the arm end point; however, the orientations of the relative coordinate frames at the end of the third link are widely different. It should be noted that wrist control variable solutions may not exist for any or all of the arm solutions; further, no examination has been made yet as to whether the control variable settings are attainable. These problems will be addressed in subsequent sections.

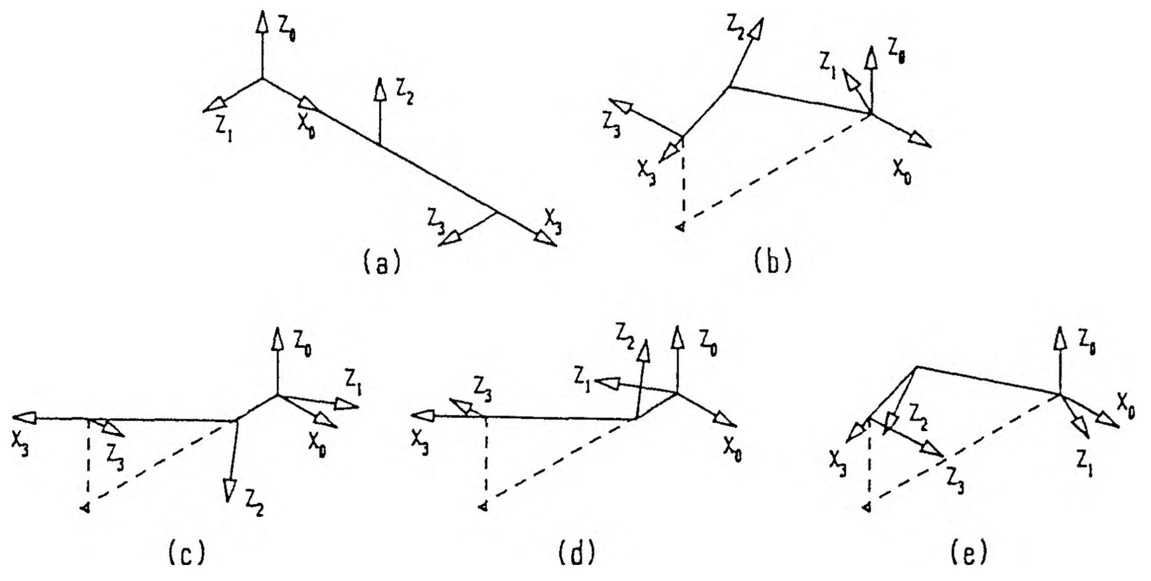


Figure 5.12. Arm Configuration for Solution Triples

3. Program Structure. The derivation of equations led first to a relationship for the third control variable, θ_3. This was followed by a series of equations involving θ_1: $\beta - \theta_1$, β, and finally θ_1 itself. Joint variable θ_2 was then generated based upon the results obtained for the previous variables. The controlling procedure of this section

follows this sequence. Procedure `calc_theta_123_triples` invokes a procedure for each of the listed steps.

```
calc_theta_3 (pa, theta, row+2, cols);
calc_beta_minus_theta_1 (pa, theta, bmt1);
beta = calc_beta (pa);
calc_theta_1 (beta, bmt1, theta, row, cols);
calc_theta_2 (pa, theta, row+1, cols);
```

Procedure `calc_theta_3` derives values for its angle, beginning by using Equations (5.81) and (5.82) to determine the cosine and sine of the angle.

```
c3 = (square (magnitude (pa)) - square (a2) - square (a3))
   / (2 * a2 * a3);
s3 = sqrt(1 - square (c3));
if ( fabs(c3) > tolerance )
      {
      theta[1][3] = atan( s3 / c3);
      theta[3][3] = atan(-s3 / c3);
                                        /* adjust atan for cos < 0 */
      if ( c3 < 0 )
         {
                                        /* c3<0, s3>0 => 4th->2nd quad */
         theta[1][3] += pi;
                                        /* c3<0, s3<0 => 1st->3rd quad */
         theta[3][3] -= pi;
         }
      }
   else
      {
      theta[1][3] =  pi / 2;
      theta[3][3] = -pi / 2;
      }
                                        /* 2 copies => 4 triples */
theta[2][3] = theta[1][3];
theta[4][3] = theta[3][3];

for (i = 1; i <= 4; i++)
   lcprintf (row, cols[i], theta[i][3] * 180/pi);
```

As will be the case with all control variables, the situation wherein the cosine is zero must be singled out for special treatment. Providing that division is possible, the arc tangent relationship is used to obtain two possible values for θ_3, as the sign of the angle's sine could not be determined. If the cosine is negative, then both of the angular results must be adjusted, from the fourth quadrant to the second and from the first to the third, where appropriate. On the other hand, if the cosine were zero, the two possibilities are positive and negative 90°. Lastly, one copy of each of the values is made to correspond to the two possibilities each for θ_1 and θ_2 which will be dependent on θ_3.

The next step is the calculation of the possibilities for the angular difference $\beta - \theta_1$. This is accomplished by procedure `calc_beta_minus_theta_1`. First the sine and positive cosine of the difference are obtained, as expressed by Equations (5.98) and (5.99), for each of the two θ_3 values.

```
                                             /* for 2 pairs:  (1,2) & (3,4) */
    for (i = 1; i <= 3; i = i + 2)
       {
       sbmt1 = a3 * sin(theta[i][3])
               / sqrt(square (pa[0]) + square (pa[1]));
       cbmt1 = sqrt(1 - square (sbmt1));
       if ( fabs(cbmt1) > tolerance )
               {
               bmt1[i  ] = atan(sbmt1 /  cbmt1);
               bmt1[i+1] = atan(sbmt1 / -cbmt1);
                                             /* adjust 1 due to sine */
               bmt1[i+1] = bmt1[i+1] + pi;
               }
          else
               {
               if ( sbmt1 > 0 )
                     bmt1[i] =  pi / 2;
                  else
                     bmt1[i] = -pi / 2;
               bmt1[i+1] = bmt1[i];
               }
       }
```

The possibility of the cosine being zero is then examined; when the cosine of the difference is not zero, there are two possibilities for each θ_3. If the sine of the difference is positive, then the resultant values must lie in the first and second quadrants; `atn(sbmt1 / -cbmt1)` would yield a fourth quadrant angle and must be adjusted. Similarly, if the sine of the difference is negative, then the results must lie in the third and fourth quadrants; in this case, `atn(sbmt1 / -cbmt1)` would produce a first quadrant angle and must be adjusted. Thus the same correction is required in either case. If on the other hand the cosine were zero, then there would only be a single resultant angle, either a positive or negative 90°. The sign of the sine determines which, and the value is used twice.

With the difference $\beta - \theta_1$ known, evaluation of β independently will subsequently lead to the value of θ_1. The procedure of the 6300 block calculates the single possible value for β, beginning with the observation made following Equations (5.102) and

(5.103) that the signs of p_{a_x} and p_{a_y} are also the signs of the cosine and sine of β, respectively.

```
    sign_cos_beta = sign (pa[0]);
    sign_sin_beta = sign (pa[1]);
    if ( fabs(pa[0]) > tolerance )
          {
          beta = atan (pa[1] / pa[0]);
                                              /* adjust atan for cos < 0 */
          if ( sign_cos_beta == -1 )
             beta += pi;
          }
       else
                                              /* cos = 0, + sin => +90 deg */
          if ( sign_sin_beta == +1 )
               beta =  pi / 2;
                                              /* cos = 0, - sin => -90 deg */
             else
               beta = -pi / 2;
    return (beta);
```

The possibility of a cosine of zero is then examined; the cosine will be zero only when the numerator of the fraction denoting the cosine in Equation (5.102) is zero. When the cosine is not zero, the arc tangent function returns a result which is adjusted for negative cosines. Equation (5.104) defines the expression for β. If the cosine is zero, then as before the sign of the sine determines the angle.

Now that $\beta - \theta_1$ and β are known, θ_1 follows by subtraction of the former from the latter. Procedure `calc_theta_1` determines the possible values for θ_1, beginning with the use of Equation (5.105), paraphrased above.

```
    for (i = 1; i <= 4; i++)
       {
                                              /* t1 = beta - (beta - t1) */
       theta[i][1] = beta - bmt1[i];
                                              /* adjust if >  180 degrees */
       if ( theta[i][1] >  pi )
          theta[i][1] -= 2 * pi;
                                              /* adjust if < -180 degrees */
       if ( theta[i][1] < -pi )
          theta[i][1] += 2 * pi;
       lcprintf (row, cols[i], theta[i][1] * 180/pi);
       }
```

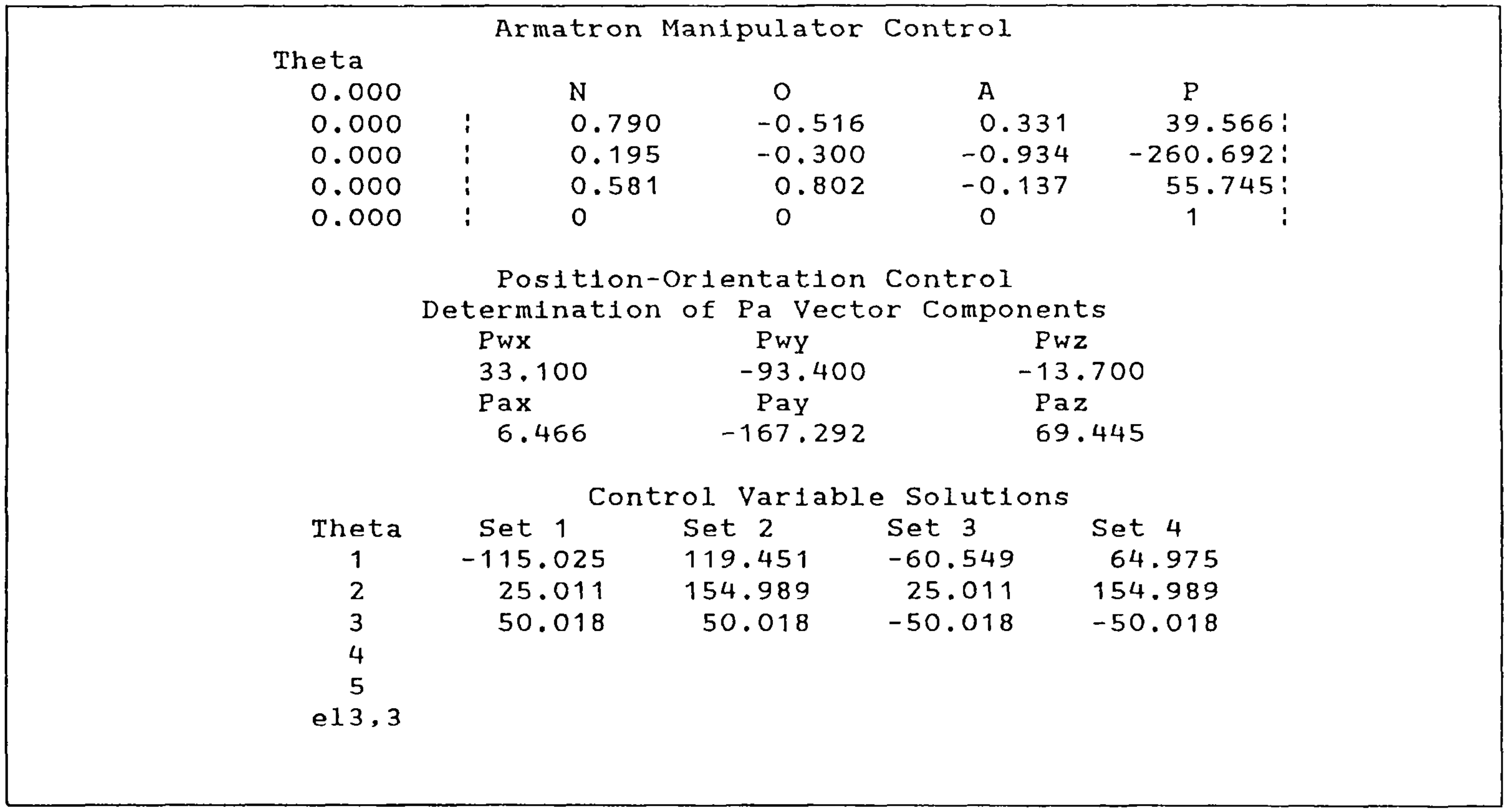
If the resultant θ_1 is greater than 180°, it is adjusted by a reduction of 360° to place it between $-180°$ and $+180°$. This is done for compliance with the joint variable restrictions; similarly, θ_1 is adjusted by the addition of 360° if it is less than $-180°$.

The third and final arm control variable, θ_2, is determined by procedure `calc_theta_2`. The procedure begins by determining the cosines and sine of previously determined variables necessary for each θ_2. Note that the relationship for c_3 in Equation (5.81) is dependent upon fixed values only, so c_3 need be calculated only once for all of the θ_2 values.

```
                                         /* cos(theta 3) is constant */
    c3 = cos(theta[1][3]);
    for (i = 1; i <= 4; i++)
      {
      c2 = ( pa[0]*cos(theta[i][1]) + pa[1]*sin(theta[i][1]) )
         / (a3*c3 + a2);
      s2 = pa[2] / (a3*c3 + a2);
      if ( fabs(c2) > tolerance )
          {
          theta[i][2] = atan(s2 / c2);
                                         /* adjust atan for cos < 0 */
          if ( c2 < 0 )
                                         /* -c, +s => 2nd quad from 4th */
             if ( s2 >= 0 )
                 theta[i][2] += pi;
                                         /* -c, -s => 3rd quad from 1st */
               else
                 theta[i][2] -= pi;
          }
        else
          if ( s2 > 0 )
               theta[i][2] =  pi / 2;
             else
               theta[i][2] = -pi / 2;
      lcprintf (row, cols[i], theta[i][2] * 180/pi);
      }
```

The cosine and sine of θ_2, given by Equations (5.113) and (5.115), respectively, are calculated next. Note that while division by zero is possible in these equations, specifically when c_3 = -1, it is not checked for in this procedure. Such an occurrence is caused by all three of the arm vector components being zero simultaneously, as described in the derivation of equations. This circumstance will be accommodated at a higher level, between calls to the arm vector component procedure and that of the arm control variables. No solution can exist in such a situation, and this procedure will not have been entered. The cosine of θ_2 can take on a value of zero; this situation is handled separately from all others. Provided the cosine is not zero, the angle is determined using the arc tangent relationship. The result of the arc tangent is adjusted in two ways, moving in the positive direction from the fourth quadrant to the second and the negative direction from the first to the third quadrant, so as to keep the resultant angle between -180° and $+180^\circ$. As usual, the angle is decided by the sign of the sine when the cosine is zero.

4. Program Example. After procedure `calc_theta_123_triples` has executed, the display contains the information shown in Figure 5.13. Note that the results are not appreciably different from those obtained in the numerical example. The differences may be attributed to the higher degree of precision to which a computer performs the involved calculations.



```
                    Armatron Manipulator Control
          Theta
            0.000          N          O          A          P
            0.000   :    0.790     -0.516      0.331     39.566:
            0.000   :    0.195     -0.300     -0.934   -260.692:
            0.000   :    0.581      0.802     -0.137     55.745:
            0.000   :    0         0          0          1     :

                    Position-Orientation Control
                Determination of Pa Vector Components
                   Pwx            Pwy            Pwz
                   33.100        -93.400        -13.700
                   Pax            Pay            Paz
                    6.466       -167.292         69.445

                         Control Variable Solutions
            Theta     Set 1      Set 2      Set 3      Set 4
              1     -115.025    119.451    -60.549     64.975
              2       25.011    154.989     25.011    154.989
              3       50.018     50.018    -50.018    -50.018
              4
              5
            el3,3
```

Figure 5.13. Display for Arm Control Variable Solutions

E. WRIST CONTROL VARIABLES

1. Derivation of Equations. With four possible combinations of arm control variables established, solutions for the wrist control variables θ_4 and θ_5 are investigated next. Recall again that link matrix i may be interpreted as transforming link i and those following it with respect to link i's coordinate frame. Let R_i be the upper 3-by-3 sub-matrix of the A_i matrix; R_i then specifies the rotation due to A_i. The matrix product $R_1R_2R_3R_4R_5$ achieves the orientation specified by the normal, orientation, and approach vectors. The sub-product R_4R_5 results in a transformation matrix which will rotate the coordinate frame at the end of the third link to that of the end-effector; the sub-product $R_1R_2R_3$ produces the transformation matrix whose columns are the normal, orientation, and approach vectors of the end of the third link. As an equation,

$$\begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} = (R_1R_2R_3)(R_4R_5) \tag{5.170}$$

The work of the previous section determined possible values for θ_1, θ_2, and θ_3. The remaining variables θ_4 and θ_5 are now solved for in Equation (5.170).

The solution process begins by isolating the matrix sub-product of the wrist link rotation matrices, R_4R_5. This is done by pre-multiplying both sides of Equation (5.170) by the inverse of $R_1R_2R_3$.

$$(R_1R_2R_3)^{-1}\begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} = (R_1R_2R_3)^{-1}(R_1R_2R_3)(R_4R_5) \tag{5.171}$$

$$(R_1R_2R_3)^{-1}\begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} = R_4R_5 \tag{5.172}$$

The product $R_1R_2R_3$ is obtained from the upper left 3-by-3 matrix of the product $A_1A_2A_3$ in Equation (4.72).

$$R_1R_2R_3 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_1s_2 & c_1c_2s_3 + s_1c_3 \\ s_1c_2c_3 + c_1s_3 & -s_1s_2 & s_1c_2s_3 - c_1c_3 \\ s_2c_3 & c_2 & s_2s_3 \end{bmatrix} \tag{5.173}$$

Recall from the preceding chapter that the original orientation vectors were simply the unit normal vector triple coincident with the base frame coordinate axes and thus may be represented in matrix form by the identity matrix; consequently, the multiplication of their matrix and any transformation by rotation results in just the transformation matrix itself. Any rotation matrix may therefore be thought of as a specification of an orientation of this unit normal vector triple. Since the columns, or vectors, of such a matrix are clearly linearly independent, the inverse of such a matrix is its own transpose. (See a linear algebra text such as [Stra80] for further explanation on this topic.) Thus the inverse of the product $R_1R_2R_3$ is its own transpose.

$$(R_1R_2R_3)^{-1} = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & s_1c_2c_3 + c_1s_3 & s_2c_3 \\ -c_1s_2 & -s_1s_2 & c_2 \\ c_1c_2s_3 + s_1c_3 & s_1c_2s_3 - c_1c_3 & s_2s_3 \end{bmatrix} \tag{5.174}$$

Substituting this inverse into Equation (5.172) yields an expression for R_4R_5.

$$R_4R_5 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & s_1c_2c_3 + c_1s_3 & s_2c_3 \\ -c_1s_2 & -s_1s_2 & c_2 \\ c_1c_2s_3 + s_1c_3 & s_1c_2s_3 - c_1c_3 & s_2s_3 \end{bmatrix} \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} \tag{5.175}$$

Now consider the matrix product R_4R_5. The matrices themselves are obtained from Equations (4.65) and (4.67), respectively, as the upper left 3-by-3 sub-matrices of the A_i matrices specified therein; their product follows.

$$R_4 = \begin{bmatrix} c_4 & 0 & s_4 \\ s_4 & 0 & -c_4 \\ 0 & 1 & 0 \end{bmatrix} \tag{5.176}$$

$$R_5 = \begin{bmatrix} c_5 & -s_5 & 0 \\ s_5 & c_5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.177}$$

$$R_4R_5 = \begin{bmatrix} c_4 & 0 & s_4 \\ s_4 & 0 & -c_4 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_5 & -s_5 & 0 \\ s_5 & c_5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.178}$$

$$R_4R_5 = \begin{bmatrix} c_4c_5 & -c_4s_5 & s_4 \\ s_4c_5 & -s_4s_5 & -c_4 \\ s_5 & c_5 & 0 \end{bmatrix} \tag{5.179}$$

Combining Equations (5.175) and (5.179),

$$\begin{bmatrix} c_4c_5 & -c_4s_5 & s_4 \\ s_4c_5 & -s_4s_5 & -c_4 \\ s_5 & c_5 & 0 \end{bmatrix} = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & s_1c_2c_3 + c_1s_3 & s_2c_3 \\ -c_1s_2 & -s_1s_2 & c_2 \\ c_1c_2s_3 + s_1c_3 & s_1c_2s_3 - c_1c_3 & s_2s_3 \end{bmatrix} \begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} \tag{5.180}$$

Instead of immediately performing the matrix product specified on the right in the equation above, note that the sines and cosines of θ_4 and θ_5 occur by themselves as elements of the matrix on the left. The row-column multiplications required for these four matrix elements are sufficient for the needed equations. Additionally, note that element (3,3) of the A_4A_5 product is zero; the corresponding row-column multiplication must also be zero.

$$a_x(c_1c_2s_3 + s_1c_3) + a_y(s_1c_2s_3 - c_1c_3) + a_z(s_2s_3) = 0 \tag{5.181}$$

If a θ_1-θ_2-θ_3 triple should fail to fit the above equation, then the triple is not viable. This may be interpreted by first considering what has been accomplished by the rotations of θ_1, θ_2, and θ_3. The matrix product $A_1A_2A_3$ results in a specific positioning of the end of the third link of the arm; in addition to this, it also causes a specific re-orientation of the vectors $\vec{n}$, $\vec{o}$, and $\vec{a}$, which were originally aligned with the base coordinate frame. To this point, these vectors have been ignored. While joint variables θ_1, θ_2, and θ_3 have accomplished the required positioning of the arm's end, it remains for joint variables θ_4 and θ_5 to achieve the required orientation of vectors $\vec{n}$, $\vec{o}$, and $\vec{a}$. Further, θ_4 and θ_5 must achieve this orientation from that left by the positioning of the end of the third link.

Now consider what can be accomplished by the rotations of the final two links of the arm. From the previous chapter, link 4 performs the θ_4 rotation about the current z-axis and then a 90° rotation about the current x-axis. Link 5 performs only the θ_5 rotation about the current z-axis, as its x-rotation is zero. Figure 5.14 describes these rotations graphically. Axes x_3, y_3, and z_3 are those at the end of the third link after the rotations of the first three links have been performed. Rotation θ_4 about axis z_3 can re-orient axis x_3 in any direction in the plane defined by x_3 and y_3 to become x_3', and y_3 is of course re-oriented accordingly to become y_3'. Figure 5.14 (1) depicts an example rotation of 65°. The 90° rotation about x_3' then brings y_3' into alignment with z_3 and $z_{3'}$ to become y_4, and z_3 re-aligns 90° away in the old x_3-y_3 plane to become z_4. Figure 5.14 (b) illustrates this turn. The final rotation θ_5 then turns axes x_4 and y_4 in their plane about axis z_4 to any new orientation to become x_5 and y_5, respectively. Figure 5.14 (c) describes an example rotation of $-35°$ for this third and final turn.

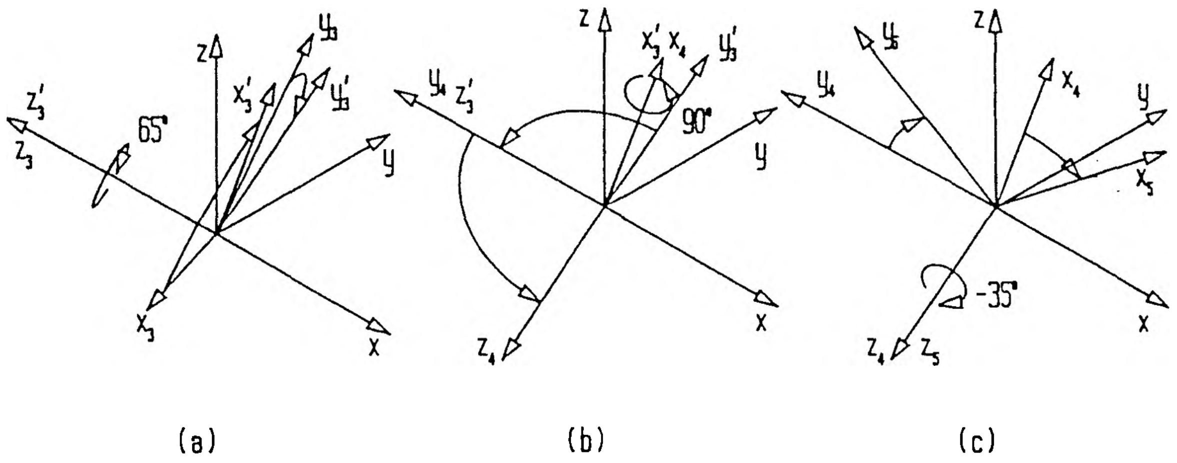


Figure 5.14. Rotation Transformations Due to Links 4 and 5

The θ_4 rotation can be thought of as lining up the x_3-axis for the x-rotation which will bring z_3 into the x_3-y_3 plane. Any direction in this plane for the resulting z_4 may be chosen by proper selection of θ_4. The final rotation θ_5 serves only to re-orient the x_4 and y_4 axes; axis z_5 remains in the x_3-y_3 plane. Therefore this sequence of rotations must result in an approach vector which is in the x-y plane of the coordinate frame at the end of the arm proper. This in turn means that the z-coordinate of the approach vector with respect to the coordinate frame of the arm's end must be zero. This is element (3,3) of the R_4R_5 matrix. If the determination of R_4R_5 results in a non-zero element for this position, the positioning of the arm's end by links 1, 2, and 3 resulted in an orientation from which the given wrist configuration's links 4 and 5 cannot achieve the final desired orientation. On the other hand, if the element is found to be zero, then the orientation is achievable, a solution does exist, and the θ_1-θ_2-θ_3 triple's corresponding θ_4 and θ_5 may be calculated.

The solutions for the wrist joint variables begin with the derivation of equations for θ_4. The cosine of θ_4 is the negative of element (2,3) in Equation (5.180) while the sine is just element (1,3).

$$-c_4 = -a_x c_1 s_2 - a_y s_1 s_2 + a_z c_2 \tag{5.182}$$

$$c_4 = a_x c_1 s_2 + a_y s_1 s_2 - a_z c_2 \tag{5.183}$$

$$s_4 = a_x(c_1 c_2 c_3 - s_1 s_3) + a_y(s_1 c_2 c_3 + c_1 s_3) + a_z s_2 c_3 \tag{5.184}$$

The results for the sine and cosine from Equations (5.183) and (5.184) are then used to determine the angle θ_4 itself.

$$\theta_4 = \tan^{-1}\left(\frac{s_4}{c_4}\right) \tag{5.185}$$

Three points should be made here. First, as before, if the cosine of the angle is negative, then the result of the arc tangent will have to be adjusted. If the sine of θ_4 is positive, then π^R is added to the above; if the sine is negative, the value is subtracted. Second, should the cosine of θ_4 be zero, one of the values $\pm\frac{\pi^R}{2}$ may be selected by examination of the sign of the sine. Lastly, it will be recalled from the previous chapter that the initial positioning of the robot arm places the gripper in a vertically downward direction to prevent a rotation about an x-axis. This in turn leaves joint variable θ_4 with the peculiar range of movement of -10° to $+190^\circ$. For this joint variable only, a special check must be made here to insure conversion of a solution in the range -180° to -170° to one in the range $+180^\circ$ to $+190^\circ$ for compliance with subsequent boundary checks.

Also from Equation (5.180), for θ_5 the cosine is element (3,2) while the sine is element (3,1).

$$c_5 = o_x(c_1c_2s_3 + s_1c_3) + o_y(s_1c_2s_3 - c_1c_3) + o_zs_2s_3 \tag{5.186}$$

$$s_5 = n_x(c_1c_2s_3 + s_1c_3) + n_y(s_1c_2s_3 - c_1c_3) + n_zs_2s_3 \tag{5.187}$$

As with θ_4, the results for the sine and cosine from Equations (5.186) and (5.187) are used to determine the angle θ_5.

$$\theta_5 = \tan^{-1}\left(\frac{s_5}{c_5}\right) \tag{5.188}$$

The remarks made concerning adjustments to the arc tangent result and handling of the zero cosine condition made for θ_4 apply also to θ_5, but the special conversion check does not as θ_5 has a range of -360° to $+360^\circ$.

2. Numerical Example. The numerical example continues by generating one θ_4-θ_5 pair for each of the θ_1-θ_2-θ_3 triples of the previous section. The process begins by checking each triple against Equation (5.181) using the approach vector components specified in Equation (5.15).

$$a_x(c_1c_2s_3 + s_1c_3) + a_y(s_1c_2s_3 - c_1c_3) + a_z(s_2s_3) = 0 \tag{5.189}$$

$$0.331(c_1c_2s_3 + s_1c_3) + (-0.934)(s_1c_2s_3 - c_1c_3) + (-0.137)(s_2s_3) = 0 \tag{5.190}$$

The first triple was $(\theta_1,\ \theta_2,\ \theta_3) = (-114.993^\circ,\ 25.027^\circ,\ 49.989^\circ)$. The left hand side of the above equation is evaluated as

$$\begin{aligned}&(0.331)(\cos(-114.993^\circ)\cos(25.027^\circ)\sin(49.989^\circ)\\&\quad+\sin(-114.993^\circ)\cos(49.989^\circ))\\&+(-0.934)(\sin(-114.993^\circ)\cos(25.027^\circ)\sin(49.989^\circ)\\&\quad-\cos(-114.993^\circ)\cos(49.989^\circ))\\&+(-0.137)(\sin(25.027^\circ)\sin(49.989^\circ))=-0.001\end{aligned} \tag{5.191}$$

This result is not significantly different from zero, so this first triple forms part of a solution and will proceed to the next step. This may be visualized as in Figure 5.15 as follows. First, the product $R_1R_2R_3$ of Equation (5.173) is evaluated for the given triple.

$$R_1R_2R_3=\begin{bmatrix} c_1c_2c_3-s_1s_3 & -c_1s_2 & c_1c_2s_3+s_1c_3\\ s_1c_2c_3+c_1s_3 & -s_1s_2 & s_1c_2s_3-c_1c_3\\ s_2c_3 & c_2 & s_2s_3\end{bmatrix} \tag{5.192}$$

$$R_1R_2R_3=\begin{bmatrix} 0.448 & 0.179 & -0.876\\ -0.852 & 0.383 & -0.357\\ 0.272 & 0.906 & 0.324\end{bmatrix} \tag{5.193}$$

This is the orientation of the vectors $\vec{n}$, $\vec{o}$, and $\vec{a}$ at the end of the third link. The desired orientation is given by the upper left 3-by-3 matrix of Equation (5.15).

$$\begin{bmatrix} n_x & o_x & a_x\\ n_y & o_y & a_y\\ n_z & o_z & a_z\end{bmatrix}=\begin{bmatrix} 0.790 & -0.516 & 0.331\\ 0.195 & -0.300 & -0.934\\ 0.581 & 0.802 & -0.137\end{bmatrix} \tag{5.194}$$

The triple of vectors specified by Equation (5.193) are labeled as $\vec{n}_3$, $\vec{o}_3$, and $\vec{a}_3$ in the figure; vector $\vec{n}_3$ is directed out of the figure and is not visible. The view in Figure 5.15 puts the plane of vectors $\vec{n}_3$ and $\vec{o}_3$ horizontal and perpendicular to the surface of the view plane. The vectors of Equation (5.194) are labeled $\vec{n}$, $\vec{o}$, and $\vec{a}$. If the wrist were to be capable of attaining the desired orientation, vector $\vec{a}$ would have to be in the plane of vectors $\vec{n}_3$ and $\vec{o}_3$. As can be seen, it is.

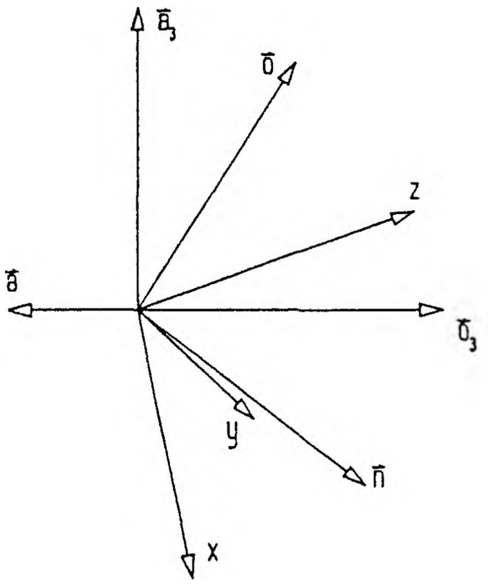


Figure 5.15. Attainable Orientation from Frame of Link 3

The second triple was $(\theta_1, \theta_2, \theta_3) = (119.419^\circ, 154.973^\circ, 49.989^\circ)$. The left hand side of Equation (5.190) is evaluated for these angles as

$$\begin{aligned}
&(0.331)(\cos(119.419^\circ)\cos(154.973^\circ)\sin(49.989^\circ) \\
&\quad + \sin(119.419^\circ)\cos(49.989^\circ)) \\
&+ (-0.934)(\sin(119.419^\circ)\cos(154.973^\circ)\sin(49.989^\circ) \\
&\quad - \cos(119.419^\circ)\cos(49.989^\circ)) \\
&+ (-0.137)(\sin(154.973^\circ)\sin(49.989^\circ)) = 0.523
\end{aligned} \tag{5.195}$$

This result is significantly different from zero, so this triple does not form part of a solution and will not proceed to the next step. This situation may be seen in Figure 5.16; the view in this figure is of the same orientation as that of Figure 5.15. The product $R_1R_2R_3$ evaluates for the second θ_1-θ_2-θ_3 triple as

$$R_1 R_2 R_3 = \begin{bmatrix} -0.381 & 0.208 & 0.901 \\ -0.884 & -0.368 & -0.289 \\ 0.272 & -0.906 & 0.324 \end{bmatrix} \qquad (5.196)$$

As before the triple of vectors specified in the above equation are labeled $\vec{n}_3$, $\vec{o}_3$, and $\vec{a}_3$ in the figure, while the final orientation vectors are labeled $\vec{n}$, $\vec{o}$, and $\vec{a}$. It may be seen that vector $\vec{a}$ does not lie in the plane of vectors $\vec{n}_3$ and $\vec{o}_3$, so a solution for θ_4 and θ_5 does not exist.

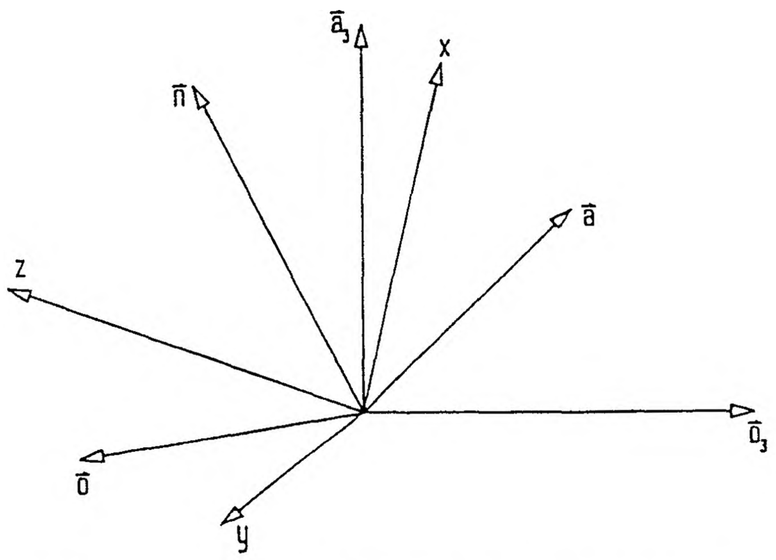


Figure 5.16. Unattainable Orientation from Frame of Link 3

The process is repeated for the third triple, (−60.581°, 25.027°, −49.989°), with a result of -0.523. Again, this value differs significantly from zero so this triple does not form part of a solution. The final triple, (65.007°, 154.973°, −49.989°), produces +0.001 as a result from evaluating the left hand side of Equation (5.190). As with the first set, this value does not differ significantly from zero so the triple will be part of a complete solution in the next step.

With the first verified triple (-114.993°, 25.027°, 49.989°), corresponding values are evaluated for first the cosines and sines of θ_4 and θ_5 and then the angles themselves. Equations (5.183) and (5.184) are employed first for θ_4 using the values of a_x, a_y, and a_z from Equation (5.15).

$$c_4 = a_x c_1 s_2 + a_y s_1 s_2 - a_z c_2 \tag{5.197}$$

$$\begin{aligned} c_4 = {} & (0.331)\cos(-114.993^\circ)\sin(25.027^\circ) \\ & + (-0.934)\sin(-114.993^\circ)\sin(25.027^\circ) - (-0.137)\cos(25.027^\circ) \end{aligned} \tag{5.198}$$

$$c_4 = 0.423 \tag{5.199}$$

$$s_4 = a_x(c_1c_2c_3 - s_1s_3) + a_y(s_1c_2c_3 + c_1s_3) + a_z s_2 c_3 \tag{5.200}$$

$$\begin{aligned} s_4 = {} & (0.331)(\cos(-114.993^\circ)\cos(25.027^\circ)\cos(49.989^\circ) \\ & \qquad - \sin(-114.993^\circ)\sin(49.989^\circ)) \\ & + (-0.934)(\sin(-114.993^\circ)\cos(25.027^\circ)\cos(49.989^\circ) \\ & \qquad + \cos(-114.993^\circ)\sin(49.989^\circ)) \\ & + (-0.137)\sin(25.027^\circ)\cos(49.989^\circ) \end{aligned} \tag{5.201}$$

$$s_4 = 0.906 \tag{5.202}$$

Equation (5.185) then gives θ_4 as

$$\theta_4 = \tan^{-1}\left(\frac{s_4}{c_4}\right) \tag{5.203}$$

$$\theta_4 = \tan^{-1}\left(\frac{0.906}{0.423}\right) = 64.973^\circ \tag{5.204}$$

Since the cosine of θ_4 is positive, the result of the arc tangent does not need to be adjusted.

Turning to θ_5, substitutions are made from Equation (5.15) for the components of the orientation and normal vectors in Equations (5.186) and (5.187) for the cosine and sine of the angle, respectively.

$$c_5 = o_x(c_1c_2s_3 + s_1c_3) + o_y(s_1c_2s_3 - c_1c_3) + o_zs_2s_3 \tag{5.205}$$

$$\begin{aligned} c_5 = (-0.516)(&\cos(-114.993^\circ)\cos(25.027^\circ)\sin(49.989^\circ) \\ &+ \sin(-114.993^\circ)\cos(49.989^\circ)) \\ + (-0.300)(&\sin(-114.993^\circ)\cos(25.027^\circ)\sin(49.989^\circ) \\ &- \cos(-114.993^\circ)\cos(49.989^\circ)) \\ + (0.802)&\sin(25.027^\circ)\sin(49.989^\circ) \end{aligned} \tag{5.206}$$

$$c_5 = 0.819 \tag{5.207}$$

$$s_5 = n_x(c_1c_2s_3 + s_1c_3) + n_y(s_1c_2s_3 - c_1c_3) + n_zs_2s_3 \tag{5.208}$$

$$\begin{aligned} s_5 = (0.790)(&\cos(-114.993^\circ)\cos(25.027^\circ)\sin(49.989^\circ) \\ &+ \sin(-114.993^\circ)\cos(49.989^\circ)) \\ + (0.195)(&\sin(-114.993^\circ)\cos(25.027^\circ)\sin(49.989^\circ) \\ &- \cos(-114.993^\circ)\cos(49.989^\circ)) \\ + (0.581)&\sin(25.027^\circ)\sin(49.989^\circ) \end{aligned} \tag{5.209}$$

$$s_5 = -0.573 \tag{5.210}$$

Equation (5.188) then gives θ_5 as

$$\theta_5 = \tan^{-1}\left(\frac{s_5}{c_5}\right) \tag{5.211}$$

$$\theta_5 = \tan^{-1}\left(\frac{-0.573}{0.819}\right) = -34.978^\circ \tag{5.212}$$

As was the case for θ_4, the result of the arc tangent does not need to be adjusted since the cosine is positive. This completes one solution for the position and orientation specified by Equation (5.15).

The above process for θ_4 and θ_5 is now repeated for the other verified θ_1-θ_2-θ_3 triple, (65.007°, 154.973°, −49.989°). Omitting the calculations for this second time through, the notable results begin with the cosine, sine, and subsequent value of θ_4.

$$c_4 = -0.423 \tag{5.213}$$

$$s_4 = 0.906 \tag{5.214}$$

$$\theta_4 = \tan^{-1}\left(\frac{0.906}{-0.423}\right) = -64.973^\circ \tag{5.215}$$

Since the cosine of the angle is negative, the result of the arc tangent must be adjusted. Since the sine of the angle is positive, 180° must be added to the above result to change the fourth quadrant arc tangent result to the desired second quadrant angle.

$$\theta_4 = -64.973^\circ + 180^\circ = 115.027^\circ \tag{5.216}$$

The results of the calculations for the cosine, sine, and subsequent value of θ_5 are next.

$$c_5 = -0.819 \tag{5.217}$$

$$s_5 = 0.573 \tag{5.218}$$

$$\theta_5 = \tan^{-1}\left(\frac{0.573}{-0.819}\right) = -34.978^\circ \tag{5.219}$$

Here as well, the result of the arc tangent needs to be adjusted since the cosine of the angle is negative. Since the sine is positive, 180° must be added to change the fourth quadrant arc tangent result above to the desired second quadrant angle.

$$\theta_5 = -34.978^\circ + 180^\circ = 145.022^\circ \tag{5.220}$$

This completes the second of the two solutions for the position and orientation originally specified by Equation (5.15). The two complete solutions for the given example are summarized in Table 5.11.

Table 5.11. SOLUTION SETS FOR ARM AND WRIST CONTROL VARIABLES

Control Variable	Set 1	Set 2
1	-114.993	65.007
2	25.027	154.973
3	49.989	-49.989
4	64.973	115.027
5	-34.978	145.022

The arm configurations resulting from each of the solution sets are depicted in Figure 5.17. Figure 5.17 (a) shows the result of the sequence of transformations dictated by the first solution set. As it turns out, the arm can achieve this solution. Figure 5.17 (b) then shows the result of the transformations of the second solution set. It should be noted that the arm will not actually be able to use this solution as the second variable, θ_2 = 154.989°, is out of the range for the second joint. Note that the position and orientation of the gripper is the same in each.

3. Program Structure. The first item to be handled by the controlling procedure of this section, `calc_theta_45_pairs`, is the determination of element (3,3) of the product of the arm inverse and orientation vector matrices for each θ_1-θ_2-θ_3 triple. This particular element must be zero or no solution can exist, as noted in the derivation

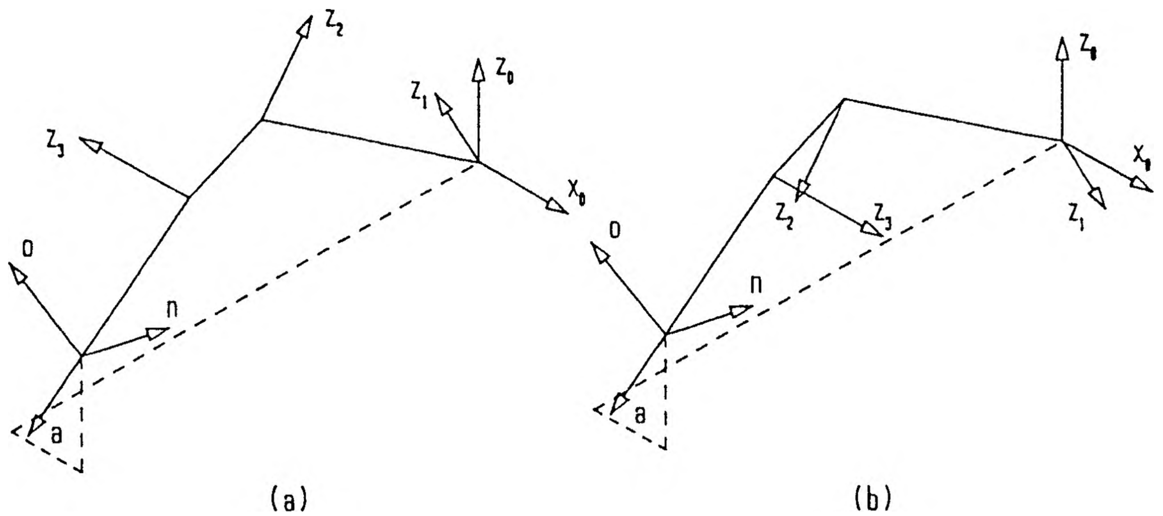


Figure 5.17. Arm and Wrist Configurations for Solution Sets

of equations. The iteration of the procedure begins by calculating the element's value for a given triple as specified by Equation (5.181).

```
atc = 0;
for (i = 1; i <= 4; i++)
   {
   for (j = 1; j <= 3; j++)
      {
      c[j] = cos(theta[i][j]);
      s[j] = sin(theta[i][j]);
      }
   el_3_3 = noap[2][0] * (c[1]*c[2]*s[3] + s[1]*c[3])
          + noap[2][1] * (s[1]*c[2]*s[3] - c[1]*c[3])
          + noap[2][2] *  s[2]*s[3];
   lcprintf (row+5, cols[i], el_3_3);
   if ( fabs(el_3_3) < tolerance )
         {
         atc++;
         for (j = 1; j <= 3; j++)
            accepted_theta[atc][j] = theta[i][j];
         accepted_theta[atc][4] = calc_theta_4 (noap[2], s, c);
         lcprintf (row+3, cols[i], accepted_theta[atc][4] * 180/pi);
         accepted_theta[atc][5] = calc_theta_5 (noap[0], noap[1],
                                                s, c);
         lcprintf (row+4, cols[i], accepted_theta[atc][5] * 180/pi);
         }
      else
         {
         lcputs (row+3, cols[i], "     No     ");
         lcputs (row+4, cols[i], "  Solution");
         }
   }
return (atc);
```

If the value is acceptably close to zero, the wrist variables are determined and the entire set is saved for use in the next section.

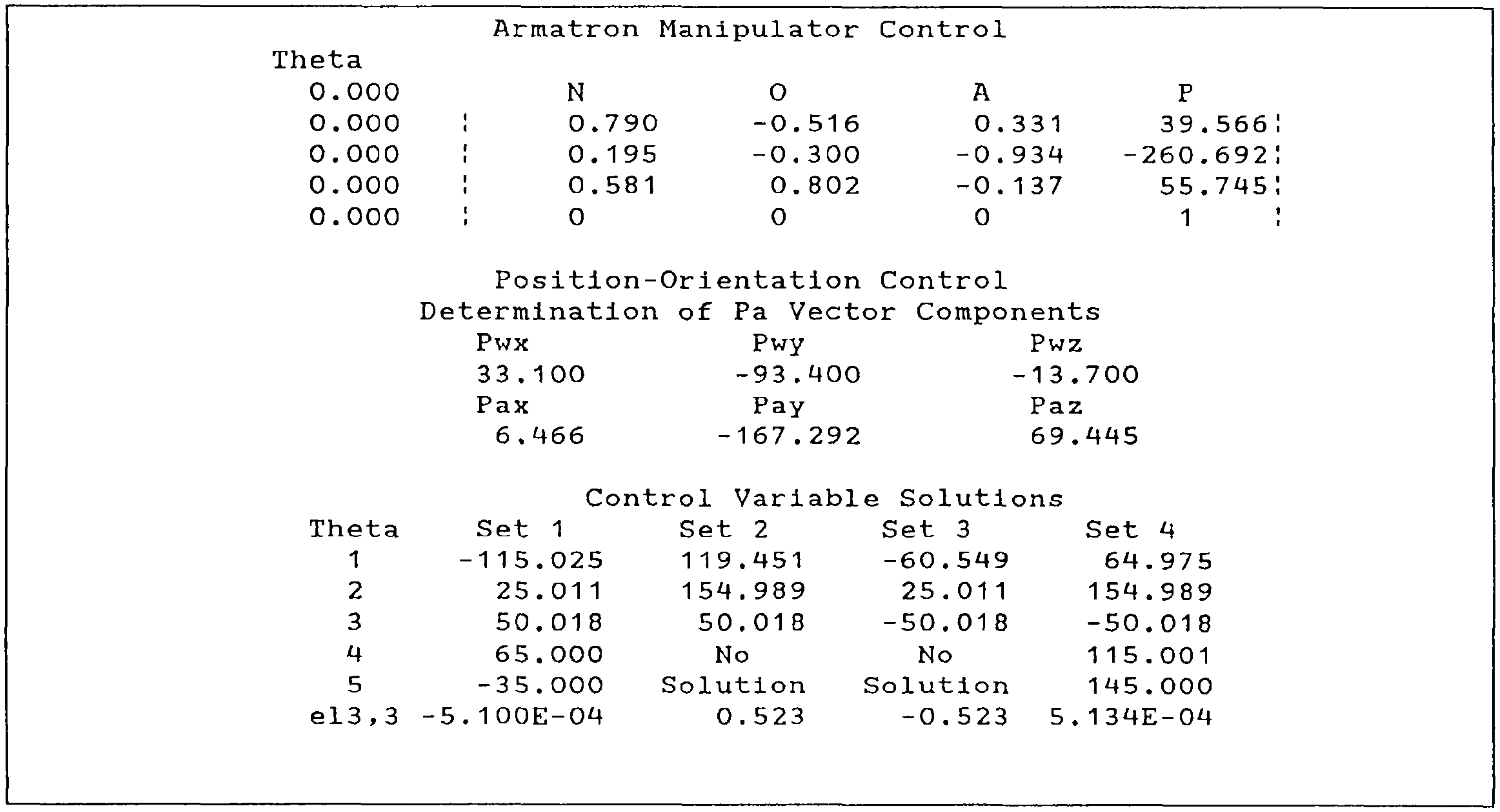
Procedure **`calc_theta_4`** determines a value for θ_4 dependent upon the current θ_1-θ_2-θ_3 triple. The process begins by determining the cosine and sine of θ_4 as given by Equations (5.183) and (5.184), respectively.

```
c4 = a[0]*c[1]*s[2] + a[1]*s[1]*s[2] - a[2]*c[2];
s4 = a[0] * (c[1]*c[2]*c[3] - s[1]*s[3])
   + a[1] * (s[1]*c[2]*c[3] + c[1]*s[3]) + a[2]*s[2]*c[3];
if ( fabs(c4) > tolerance )
       {
       theta4 = atan(s4 / c4);
                                            /* adjust atan for cos < 0 */
       if ( c4 < 0 )
                                            /* -c, +s => 2nd quad from 4th */
          if (s4 >= 0)
                theta4 += pi;
                                            /* -c, -s => 3rd quad from 1st */
             else
                theta4 -= pi;
                                            /* conv for bounds compliance */
if ( theta4 < -170 * pi/180 )
       theta4 += 2 * pi;
       }
   else
                                            /* cos = 0, +sin => +90 deg */
       if ( s4 > 0 )
             theta4 =  pi / 2;
                                            /* cos = 0, -sin => -90 deg */
          else
             theta4 = -pi / 2;
return (theta4);
```

The possibility of a zero cosine is investigated next; the angle is then determined using the arc tangent relationship. As with previous control variables, the adjustments made for a negative cosine are designed to place the angle in the range -180° to $+180^\circ$. The derivation of equations noted that the unusual boundaries for θ_4, -10° and $+190^\circ$, were due to the arm's initial alignment. A check must be made here to convert any solution in the range -180° to -170° to one in the range $+180^\circ$ to $+190^\circ$. This will allow simple comparisons to be performed when checking solution sets against the variable ranges. An angle in the range -180° to -170° will have a negative cosine, so the check may be performed following the previous arc tangent adjustments. Further, since the arc tangent adjustments above will result in a solution between -180° and $+180^\circ$, the condition in question can be detected by a single comparison with -170°. In the event of a cosine of zero, as with previous control variables the sign of the sine determines the angle when the cosine is zero.

The procedure for determining θ_5, `calc_theta_5`, parallels that of the previous control variable, θ_4, for the most part. The distinguishable differences are the use of Equations (5.186) and (5.187) for the calculation of the cosine and sine, respectively, of θ_5 in place of those for θ_4, and the deletion of the special test for a θ_4 solution in the range -180° to -170°, as θ_5 has balanced boundaries.

4. Program Example. Execution of procedure `calc_theta_45_pairs` generates the information displayed in Figure 5.18. As with the arm control variables, the solutions shown here for the wrist control variables are not appreciably different from those obtained in the numerical example.



```
                 Armatron Manipulator Control
      Theta
        0.000            N          O          A          P
        0.000    ¦     0.790     -0.516      0.331     39.566¦
        0.000    ¦     0.195     -0.300     -0.934   -260.692¦
        0.000    ¦     0.581      0.802     -0.137     55.745¦
        0.000    ¦     0           0          0          1    ¦

                 Position-Orientation Control
             Determination of Pa Vector Components
                Pwx             Pwy             Pwz
                33.100         -93.400         -13.700
                Pax             Pay             Paz
                 6.466        -167.292         69.445

                      Control Variable Solutions
        Theta     Set 1      Set 2      Set 3      Set 4
          1     -115.025    119.451    -60.549     64.975
          2       25.011    154.989     25.011    154.989
          3       50.018     50.018    -50.018    -50.018
          4       65.000       No         No      115.001
          5      -35.000   Solution   Solution    145.000
        el3,3  -5.100E-04     0.523     -0.523  5.134E-04
```

Figure 5.18. Display for Arm and Wrist Control Variable Solutions

F. VERIFICATION OF RANGES AND SELECTION OF A SOLUTION

The next phase of the problem is to take the solutions obtained by the work of the previous sections and provide one set, if one exists, whose angles are all within the stated ranges for the control variables to the robot movement routines of the next section. The user is allowed to choose from among multiple solutions or to prevent any movement at all. This procedure, **prompt_for_move**, is invoked by an upper level procedure which will examine the flag **move** for a 'Y' value before a move will be attempted. The procedure begins by displaying the accepted manipulator solutions and comparing them against their bounds, maintaining a count of how many and an index array of which solution sets are attainable.

```
dsply_prompt_for_move (&row, cols);
inbounds = 0;
for (i = 1; i <= accepted; i++)
   {
   out[i] = 0;
   for (j = 1; j <= 5; j++)
      {
      accepted_theta[i][j] *= (180 / pi);
      lcprintf (row+j-1, cols[i], accepted_theta[i][j]);
      if ( (accepted_theta[i][j] >= min_constraint (j))
         & (accepted_theta[i][j] <= max_constraint (j)) )
            lcputs (row+j-1, cols[i+4]+2, "In");
         else
            {
            lcputs (row+j-1, cols[i+4]+1, "Out");
            out[i]++;
            }
      }
   if ( out[i] == 0 )
      {
      inbounds++;
      inbounds_index[inbounds] = i;
      }
   }
switch (inbounds)
   {
   case 0 : lcputs (20, 20, "No Solution is Obtainable");
            move = 'N';
            break;
   case 1 : move = one_solution (inbounds_index[1], accepted_theta,
                                 move_theta);
            break;
   default: move = multiple_solutions (accepted, accepted_theta,
                                       inbounds, inbounds_index,
                                       out, move_theta);
   }
erase_prompt (22);
return (move);
```

The procedure concludes by acting on the number of solutions available.

Procedure **one_solution** begins by querying about an attempt to achieve the single obtained solution.

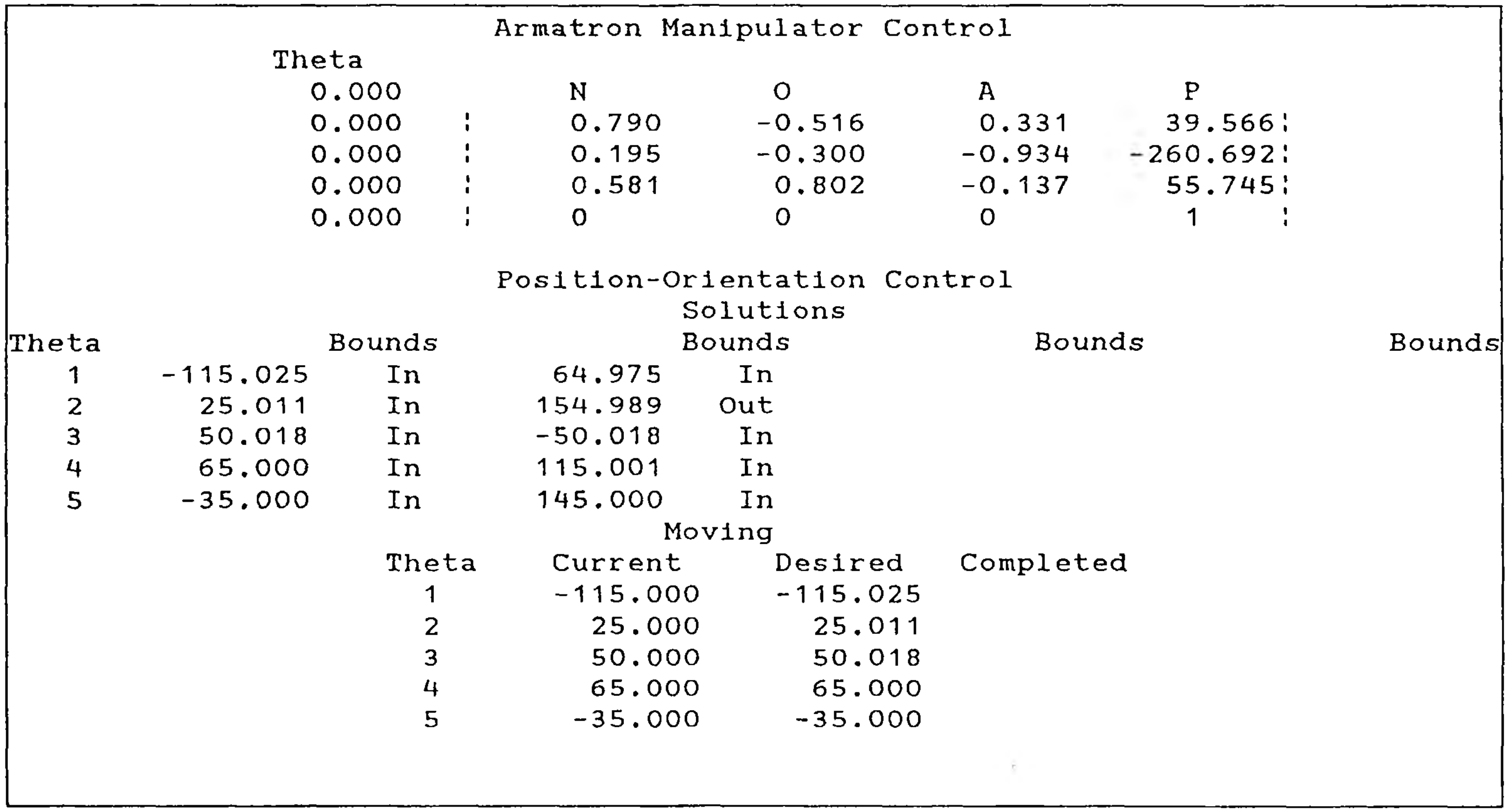
```
locate (23, 20);
cprintf ("Solution %d is obtainable", index);
lcputs (24, 20, "Perform move? (y/n)");
locate (24, 42);
move = toupper(getch( ));
if (move == 'Y')
   for (j = 1; j <= 5; j++)
      move_theta[j] = accepted_theta[index][j];
return (move);
```

The procedure concludes by setting up the array **move_theta** for use by the next section if a move is desired.

When more than one solution is detected, procedure **multiple_solutions** is invoked; it begins by listing the achievable solution subscripts and querying as to which to use.

```
locate (22, 20);
cprintf ("Solutions %d", inbounds_index[1]);
for (i = 2; i <= inbounds-1; i++)
   cprintf (", %d", inbounds_index[i]);
cprintf (" and %d are obtainable.", inbounds_index[inbounds]);
prompt = "Select set for a move or 0 to abort:";
do
   {
   set = prompt_input_digit (prompt);
   if ( set == 0 )
         cont = 0;
      else
         if ( (set <= accepted) & (out[set] == 0) )
               cont = 0;
            else
               cont = 1;
   }
   while ( cont );
locate (24, 20);
cprintf ("Solution %d has been selected", set);
if ( set == 0 )
      move = 'N';
   else
      {
      move = 'Y';
      for (j = 1; j <= 5; j++)
         move_theta[j] = accepted_theta[set][j];
      }
return (move);
```

The procedure concludes by initializing the array **move_theta** with the selected solution set for use by the next section if a move is desired. For the continuing example, execution of procedure **prompt_for_move** displays the information of Figure 5.19.



```
                        Armatron Manipulator Control
             Theta
               0.000           N          O          A          P
               0.000   :     0.790     -0.516      0.331      39.566:
               0.000   :     0.195     -0.300     -0.934    -260.692:
               0.000   :     0.581      0.802     -0.137      55.745:
               0.000   :     0         0          0          1      :

                        Position-Orientation Control
                                 Solutions
Theta               Bounds               Bounds              Bounds              Bounds
   1     -115.025     In        64.975     In
   2       25.011     In       154.989    Out
   3       50.018     In       -50.018     In
   4       65.000     In       115.001     In
   5      -35.000     In       145.000     In
                                        Moving
                        Theta     Current      Desired    Completed
                          1      -115.000     -115.025
                          2        25.000       25.011
                          3        50.000       50.018
                          4        65.000       65.000
                          5       -35.000      -35.000
```

Figure 5.19. Display for Range Verification of Solutions

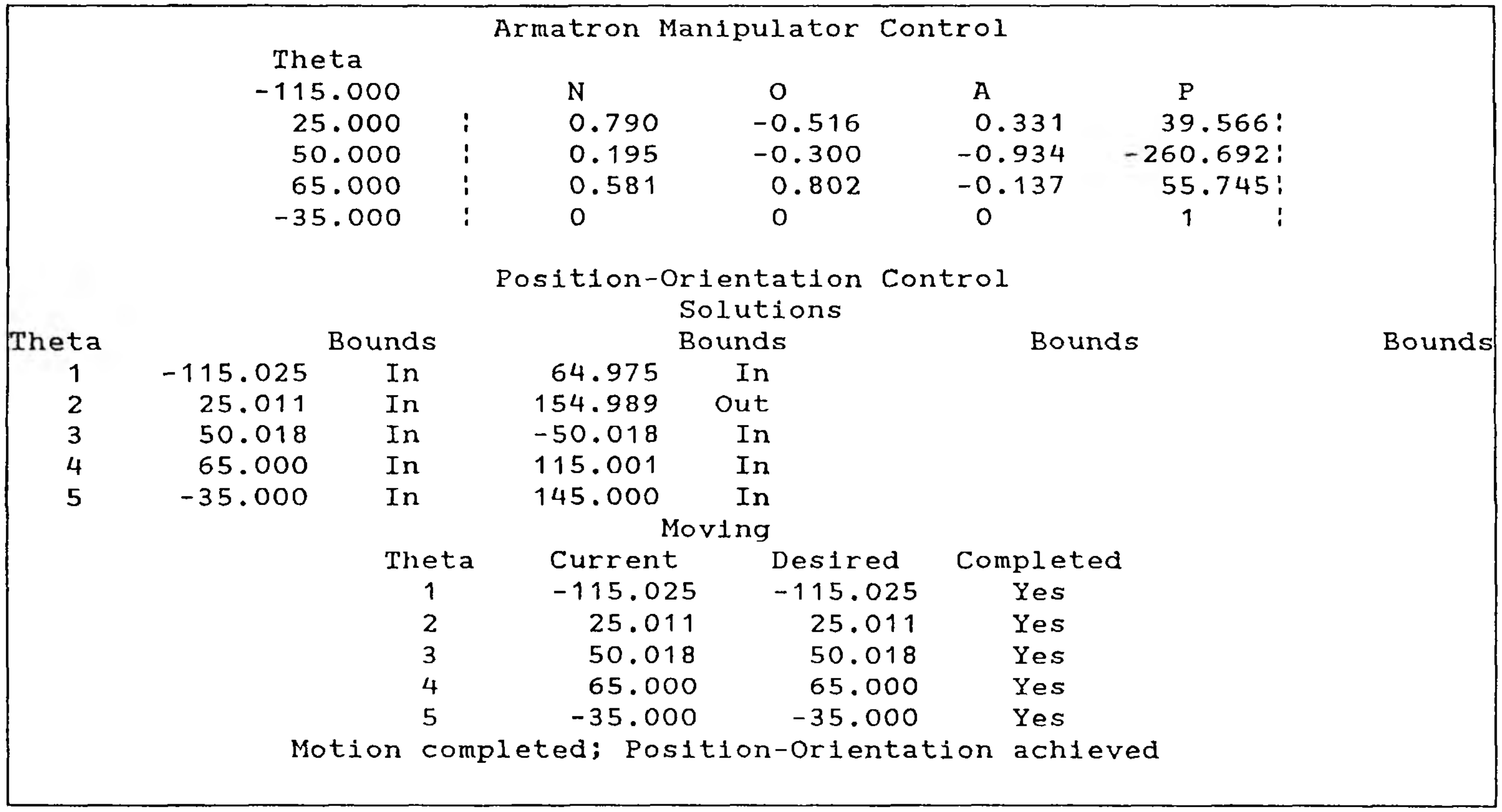
G. MOVEMENT TO THE SPECIFIED POSITION AND ORIENTATION

If a single solution were found and selected and the decision made to move the robot arm, then the procedure of this section will be invoked. Procedure `position_orientation_move` directs control for the arm movement; it begins by displaying the current settings and those desired for the manipulator's joint variables.

```
dsply_pos_orient_move (&row, cols);
for (i = 1; i <= 5; i++)
   {
   lcprintf (row+i-1, cols[0], theta[i]);
   lcprintf (row+i-1, cols[1], move_theta[i]);
   }
wait_then_erase (24);
interrupt_count = 0;
for (i = 1; i <= 5; i++)
   {
   perform_move (i, theta, move_theta[i], row+i-1, cols[0]);
   if ( fabs(theta[i] - move_theta[i]) < tolerance )
         lcputs (row+i-1, cols[2], "Yes");
      else
         {
         lcputs (row+i-1, cols[2], " No");
         interrupt_count++;
         }
   }
if ( interrupt_count == 0 )
      {
      lcputs (23, 15, "Motion completed; ");
      cputs ("Position-Orientation achieved");
      }
   else
      {
      lcputs (23, 15, "Some motion interrupted; ");
      cputs ("Position-Orientation not achieved");
      }
wait_then_erase (9);
```

As each movement is performed, the user will have the ability to stop the motion in the event that an object in the arm's envelope is interfering with the motion or a motor stalls. An interrupt count will be maintained to indicate how many of the arm's five motions were interrupted by the user. This count will indicate that the desired position and orientation were achieved if it remains zero after initialization. The procedure iterates for each of the five joint variables, invoking procedure `perform_move` of the previous chapter to perform the actual movement. Lastly, the procedure displays a completion status based upon the interruption count.

Execution of procedure **`position_orientation_move`** displays the information of Figure 5.20. Note the acknowledgement for position achieval for each individual joint in the figure.



```
                          Armatron Manipulator Control
             Theta
           -115.000            N          O          A          P
             25.000   :      0.790     -0.516      0.331      39.566:
             50.000   :      0.195     -0.300     -0.934    -260.692:
             65.000   :      0.581      0.802     -0.137      55.745:
            -35.000   :      0              0          0          1    :

                          Position-Orientation Control
                                   Solutions
Theta              Bounds               Bounds                 Bounds                 Bounds
  1     -115.025     In        64.975     In
  2       25.011     In       154.989    Out
  3       50.018     In       -50.018     In
  4       65.000     In       115.001     In
  5      -35.000     In       145.000     In
                                    Moving
                    Theta     Current      Desired     Completed
                      1      -115.025     -115.025        Yes
                      2        25.011       25.011        Yes
                      3        50.018       50.018        Yes
                      4        65.000       65.000        Yes
                      5       -35.000      -35.000        Yes
              Motion completed; Position-Orientation achieved
```

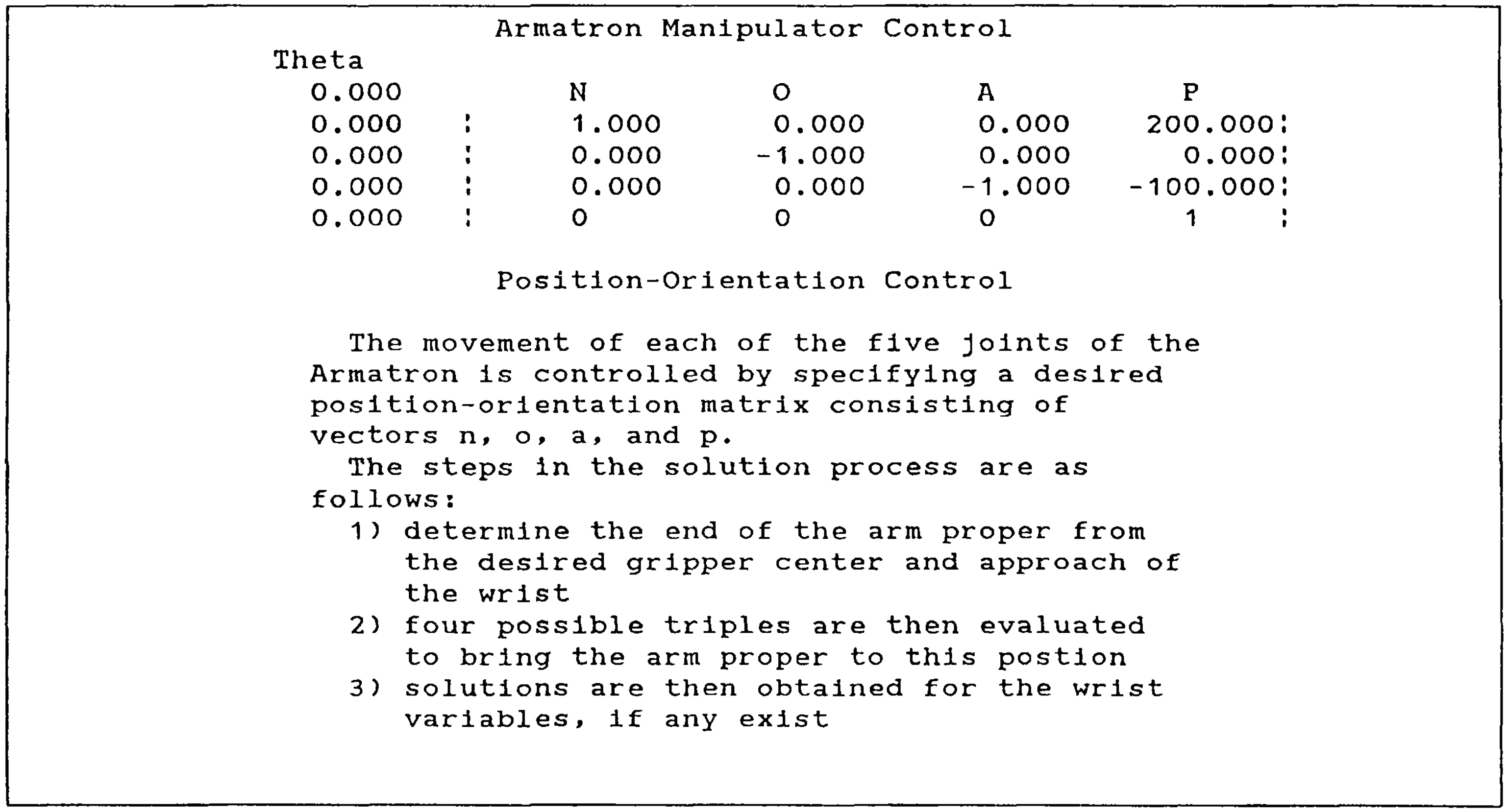
Figure 5.20. Display for Movement via a Solution

II. THE CONTROLLING PROCEDURE

Procedure **position_orientation_control** begins by generating the introductory display of Figure 5.21. It subsequently directs the execution of the procedural steps outlined in this chapter. The procedure repeats the position and orientation solution process for different matrix inputs until instructed to stop.

```
dsply_position_orientation_introduction ( );
wait_then_erase (9);
do
   {
   dsply_pos_orient_solution (&arm_row, arm_cols,
                              &theta_row, theta_cols);
   get_noap (noap, noap_row, noap_cols);
   magnitude_ok = calc_arm_end (noap, pa, arm_row, arm_cols);
   if ( magnitude_ok )
         {
         calc_theta_123_triples (pa, theta123,
                                 theta_row, theta_cols);
         accepted = calc_theta_45_pairs (noap, theta123,
                                         accepted_theta,
                                         theta_row, theta_cols);
         wait_then_erase (9);
         move = prompt_for_move (accepted, accepted_theta,
                                 move_theta);
         if ( move == 'Y' )
            position_orientation_move (theta, move_theta);
         noap_matrix (theta, noap, noap_row, noap_cols);
         }
      else
         lcputs (28, 19, "Arm end position not attainable");
   prompt_msg1 = "Continue with another N-O-A-P matrix (y/n)?";
   prompt_msg2 = "";
   qc = prompt_input_char (prompt_msg1, prompt_msg2);
   }
   while ( qc == 'Y' );
wait_then_erase (8);
```

Note that the examination is made of the end of the arm proper resulting from the specified manipulator end frame position and orientation at this level to prevent solution attempts for unattainable arm end positions. The documented listing for the procedures associated with the position and orientation control portion of the overall program may be found in Appendix E.



```
                    Armatron Manipulator Control
       Theta
        0.000          N          O          A          P
        0.000   :    1.000      0.000      0.000    200.000:
        0.000   :    0.000     -1.000      0.000      0.000:
        0.000   :    0.000      0.000     -1.000   -100.000:
        0.000   :    0                      0          1     :

                    Position-Orientation Control

          The movement of each of the five joints of the
        Armatron is controlled by specifying a desired
        position-orientation matrix consisting of
        vectors n, o, a, and p.
          The steps in the solution process are as
        follows:
          1) determine the end of the arm proper from
             the desired gripper center and approach of
             the wrist
          2) four possible triples are then evaluated
             to bring the arm proper to this postion
          3) solutions are then obtained for the wrist
             variables, if any exist
```

Figure 5.21. Position and Orientation Introductory Display

VI. VELOCITY CONTROL

The relationship between the joint control variables and the orientation and position of the coordinate frame of the arm end has been examined extensively in the two previous chapters. Specifically, Chapter 4 obtained the orientation and position of the end coordinate frame given the joint control variables, while Chapter 5 derived the more difficult reverse relationships of joint variables from a desired orientation and position of the manipulator end frame. This chapter builds upon these relationships by deriving the correspondences between the rates of change in the joint variables and the end coordinate frame. This will be done in both directions: the rate of change in the manipulator end coordinate frame's position and orientation will be developed from the rates of change in the arm control variables, and the control variable rates will likewise be derived from the rates of change in arm end orientation. In particular, the latter will allow for the determination of control variable speeds to achieve desired rates in the arm end coordinate frame.

It should be noted here that with the existing arm construction (in particular, the motors used), it will not be possible to implement directly the results obtained in this chapter. Changes in the arm design which would make this possible will be proposed in Chapter 8. It is also worth noting that the computations involved are of sufficient complexity to preclude real-time generation at any rate; thus the work done in this chapter would be accomplished off-line, independent of and prior to actual robot control. As was done for the previous chapters, the program will be developed in stages following the derivation of equations and numerical example of each section.

A. TRANSLATIONAL AND ROTATIONAL RATES OF THE END FRAME

This section provides a derivation for the rates of the manipulator end coordinate frame in terms of the joint rates. This method shall result in rates which are with respect to the base coordinate directions. It is important to distinguish here between base coordinate directions and the base coordinate frame itself. As will be seen later, a rotational rate around a base coordinate frame axis has both rotational and translational effects with respect to the manipulator end. The derivation of this section will yield the rates of the manipulator end with respect to a coordinate system centered at the manipulator end but identical in orientation to the base frame.

1. Derivation of Equations. Consider the effect the rate at which one joint transforms has on the coordinate frame at the end of the manipulator. In Chapter 4, a link was defined for the physical transformations from each joint coordinate frame to the next. When joint i rotates, link i rotates coordinate frame i about joint i's coordinate frame $i-1$. Coordinate frames $i+1$, $i+2$, ...6, are also rotated about frame $i-1$. These frames remain fixed with respect to frame i. The effect of the rotation of joint i may be broken down into translational and rotational components.

a. Translational Velocities. Consider Figure 6.1 and the three coordinate frames which are related to each other by vectors. For the equations that follow, each of the vectors shall be specified with respect to frame $i-1$. The distance and direction of vector ${}^{i-1}\vec{d}_i$ relates frame $i-1$ to frame i. The remaining distance and direction from frame i to frame n is represented by vector ${}^{i-1}\vec{d}_r$. Finally, vector ${}^{i-1}\vec{d}$ is that relating frame $i-1$ to frame n directly. Note that vector ${}^{i-1}\vec{d}$ is the vector sum of vectors ${}^{i-1}\vec{d}_i$ and ${}^{i-1}\vec{d}_r$.

$$ {}^{i-1}\vec{d} = {}^{i-1}\vec{d}_i + {}^{i-1}\vec{d}_r \qquad (6.1) $$

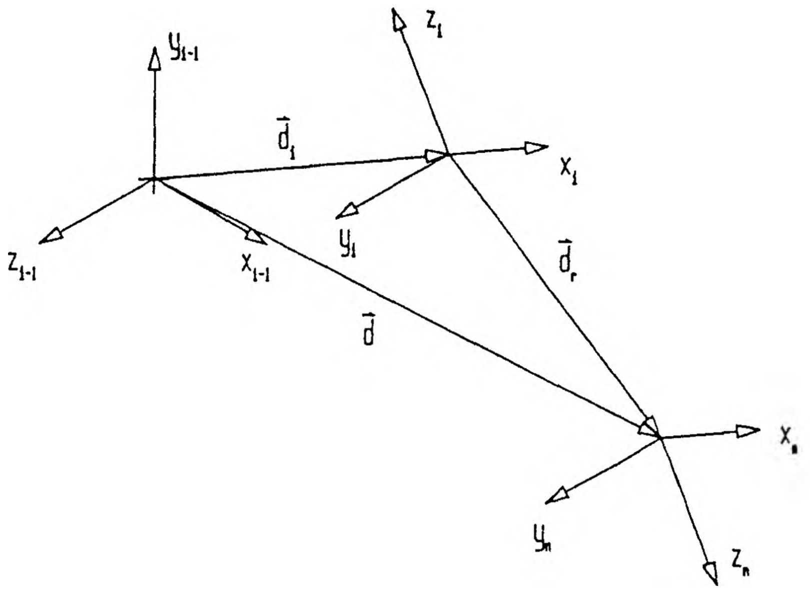


Figure 6.1. Single Joint Under Transformation

The linear velocities at frame n are obtained from this equation. The process begins by differentiating Equation (6.1) with respect to time.

$$\frac{d}{dt}\,{}^{i-1}\vec{d} = \frac{d}{dt}\,{}^{i-1}\vec{d}_i + \frac{d}{dt}\,{}^{i-1}\vec{d}_r \tag{6.2}$$

The left hand side of this equation is the desired rate of change, that of the manipulator end. The first term on the right hand side is obtained by recalling that the displacement of a given link i is specified by a rotation θ_i about axis z_{i-1}, translation of a distance d_i along axis z_{i-1}, translation along axis x_{i-1} of distance a_i, and a rotation of α_i about x_{i-1} from the direction of z_{i-1} to that of z_i. The combined displacement of these transformations, and hence vector $\vec{x}_i$, may be obtained from the components of the fourth column of the A_i matrix in Equation (4.55); the required derivative follows.

$$A_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

$$^{i-1}\vec{d}_i = \begin{bmatrix} a_i \cos\theta_i \\ a_i \sin\theta_i \\ d_i \end{bmatrix} \tag{6.4}$$

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_i = \begin{bmatrix} -a_i \sin\theta_i \frac{d}{dt}\theta_i \\ a_i \cos\theta_i \frac{d}{dt}\theta_i \\ \frac{d}{dt} d_i \end{bmatrix} \tag{6.5}$$

Note that the rate of change of d_i is that of the distance between coordinate frames $i-1$ and i, while the rate of change of $^{i-1}\vec{d}_i$ is that of the vector components with respect to coordinate frame $i-1$. The second term on the right hand side of Equation (6.2) is obtained by first relating vector $\vec{d}_r$ with respect to two different coordinate frames, i and $i-1$. This is done by referring again to the link transformation matrix in Equation (6.3) and employing here the upper left 3x3 rotational portion of the matrix; this is denoted R_i.

$$^{i-1}R_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i \\ 0 & \sin\alpha_i & \cos\alpha_i \end{bmatrix} \tag{6.6}$$

$$^{i-1}\vec{d}_r = ({}^{i-1}R_i)({}^{i}\vec{d}_r) \tag{6.7}$$

Differentiating the product of Equation (6.7),

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = \left(\frac{d}{dt}\,{}^{i-1}R_i\right){}^{i}\vec{d}_r + {}^{i-1}R_i\left(\frac{d}{dt}\,{}^{i}\vec{d}_r\right) \tag{6.8}$$

As all of the other manipulator joints are being held constant, the rate of change of ${}^{i}\vec{d}_r$ is zero and the second term in Equation (6.8) becomes zero. The first term is expanded by differentiating matrix ${}^{i-1}R_i$. Note that in doing so, only θ_i has a rate of change as α_i is assumed constant for the link; this is the case for each joint of the Armatron manipulator.

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = \left(\frac{d}{dt}\,{}^{i-1}R_i\right){}^{i}\vec{d}_r \tag{6.9}$$

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = \frac{d}{dt}\begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i \\ 0 & \sin\alpha_i & \cos\alpha_i \end{bmatrix}{}^{i}\vec{d}_r \tag{6.10}$$

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = \begin{bmatrix} -\sin\theta_i & -\cos\theta_i\cos\alpha_i & \cos\theta_i\sin\alpha_i \\ \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i \\ 0 & 0 & 0 \end{bmatrix}\left(\frac{d}{dt}\theta_i\right){}^{i}\vec{d}_r \tag{6.11}$$

This expression may be simplified by post-multiplying the ${}^{i-1}R_i$ matrix derivative by $I = ({}^{i-1}R_i)^{-1}({}^{i-1}R_i)$. The inverse of ${}^{i-1}R_i$ is simply its transpose because the columns of ${}^{i-1}R_i$ are independent of one another, as discussed in Chapter 4.

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = \begin{bmatrix} -s\theta_i & -c\theta_i c\alpha_i & c\theta_i s\alpha_i \\ c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i \\ 0 & 0 & 0 \end{bmatrix}({}^{i-1}R_i)^T({}^{i-1}R_i)\left(\frac{d}{dt}\theta_i\right){}^{i}\vec{d}_r \tag{6.12}$$

Rearranging,

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = \begin{bmatrix} -s\theta_i & -c\theta_i c\alpha_i & c\theta_i s\alpha_i \\ c\theta_i & -s\theta_i c\alpha_i & s\theta_i s\alpha_i \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} c\theta_i & s\theta_i & 0 \\ -s\theta_i c\alpha_i & c\theta_i c\alpha_i & s\alpha_i \\ s\theta_i s\alpha_i & -c\theta_i s\alpha_i & c\alpha_i \end{bmatrix}({}^{i-1}R_i){}^{i}\vec{d}_r\frac{d}{dt}\theta_i \tag{6.13}$$

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = \begin{bmatrix} -s\theta_i c\theta_i + c\theta_i s\theta_i c^2\alpha_i + c\theta_i s\theta_i s^2\alpha_i & -s^2\theta_i - c^2\theta_i c^2\alpha_i - c_2\theta_i s^2\alpha_i & -c\theta_i c\alpha_i s\alpha_i + c\theta_i c\alpha_i s\alpha_i \\ c^2\theta_i + s^2\theta_i c^2\alpha_i + s^2\theta_i s^2\alpha_i & c\theta_i s\theta_i - c\theta_i s\theta_i c^2\alpha_i - c\theta_i s\theta_i s^2\alpha_i & -s\theta_i c\alpha_i s\alpha_i + s\theta_i c\alpha_i s\alpha_i \\ 0 & 0 & 0 \end{bmatrix} ({}^{i-1}R_i)\,{}^{i}\vec{d}_r \frac{d}{dt}\theta_i \tag{6.14}$$

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = \begin{bmatrix} -s\theta_i c\theta_i(1 - c^2\alpha_i - s^2\alpha_i) & -s^2\theta_i - c^2\theta_i(c^2\alpha_i + s^2\alpha_i) & 0 \\ c^2\theta_i + s^2\theta_i(c^2\alpha_i + s^2\alpha_i) & c\theta_i s\theta_i(1 - c^2\alpha_i - s^2\alpha_i) & 0 \\ 0 & 0 & 0 \end{bmatrix} ({}^{i-1}R_i)\,{}^{i}\vec{d}_r \frac{d}{dt}\theta_i \tag{6.15}$$

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} ({}^{i-1}R_i)\,{}^{i}\vec{d}_r \frac{d}{dt}\theta_i \tag{6.16}$$

The product $({}^{i-1}R_i)\,{}^{i}\vec{d}_r$ is just ${}^{i-1}\vec{d}_r$, as stated in Equation (6.7).

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} {}^{i-1}\vec{d}_r \frac{d}{dt}\theta_i \tag{6.17}$$

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} {}^{i-1}\begin{bmatrix} d_{r_x} \\ d_{r_y} \\ d_{r_z} \end{bmatrix} \frac{d}{dt}\theta_i \tag{6.18}$$

$$\frac{d}{dt}\,{}^{i-1}\vec{d}_r = {}^{i-1}\begin{bmatrix} -d_{r_y} \\ d_{r_x} \\ 0 \end{bmatrix} \frac{d}{dt}\theta_i \tag{6.19}$$

This equation and Equation (6.5) are substituted back into Equation (6.2) to obtain the desired rate of change.

$$\frac{d}{dt}\,{}^{i-1}\vec{d} = \frac{d}{dt}\,{}^{i-1}\vec{d}_i + \frac{d}{dt}\,{}^{i-1}\vec{d}_r \tag{6.20}$$

$$\frac{d}{dt}\,{}^{i-1}\vec{d} = \begin{bmatrix} -a_i \sin\theta_i \frac{d}{dt}\theta_i \\ a_i \cos\theta_i \frac{d}{dt}\theta_i \\ \frac{d}{dt} d_i \end{bmatrix} + {}^{i-1}\begin{bmatrix} -d_{r_y} \\ d_{r_x} \\ 0 \end{bmatrix} \frac{d}{dt}\theta_i \tag{6.21}$$

Examination of the A_i matrix in Equation (6.3) shows that the product $a_i \cos\theta_i$ is the x-component with respect to frame $i-1$ of the translation vector of link i, while $a_i \sin\theta_i$ is its y-component. This translation vector is labeled $\vec{d}_i$ in Figure (6.1).

$$\frac{d}{dt}\,{}^{i-1}\vec{d} = \begin{bmatrix} {}^{i-1}{-d_{i_y}} \frac{d}{dt}\theta_i \\ {}^{i-1}d_{i_x} \frac{d}{dt}\theta_i \\ \frac{d}{dt} d_i \end{bmatrix} + {}^{i-1}\begin{bmatrix} -d_{r_y} \\ d_{r_x} \\ 0 \end{bmatrix}_r \frac{d}{dt}\theta_i \tag{6.22}$$

$$\frac{d}{dt}\,{}^{i-1}\vec{d} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \frac{d}{dt} d_i + {}^{i-1}\begin{bmatrix} -d_{i_y} - d_{r_y} \\ d_{i_x} + d_{r_x} \\ 0 \end{bmatrix} \frac{d}{dt}\theta_i \tag{6.23}$$

From Equation (6.1), the sum of vectors $\vec{d}_i$ and $\vec{d}_r$ was stated to be vector $\vec{d}$. The component sums in Equation (6.23) are replaced using this relationship.

$$\frac{d}{dt}\,{}^{i-1}\vec{d} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \frac{d}{dt} d_i + {}^{i-1}\begin{bmatrix} -d_y \\ d_x \\ 0 \end{bmatrix} \frac{d}{dt}\theta_i \tag{6.24}$$

The components of vector ${}^{i-1}\vec{d}$ may be obtained from the fourth column of the matrix product $A_i A_{i+1} \ldots A_5$, as this column represents the translation due to the involved transformations. The product of A matrices was seen in Chapter 4 to have as its columns the vectors $\vec{n}$, $\vec{o}$, $\vec{a}$, and $\vec{p}$. For the product $A_i A_{i+1} \ldots A_5$, the fourth column components are thus elements of vector ${}^{i-1}\vec{p}_5$.

$$\frac{d}{dt}\,{}^{i-1}\vec{d} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \frac{d}{dt} d_i + {}^{i-1}\begin{bmatrix} -p_y \\ p_x \\ 0 \end{bmatrix}_5 \frac{d}{dt}\theta_i \tag{6.25}$$

The rate of change of $\vec{d}$ in this equation is with respect to coordinate frame $i-1$. The equation is generalized by obtaining the rate of change with respect to the base coordinate frame. This is accomplished by premultiplying both sides by the matrix product $R_1 R_2 \ldots R_{i-1}$, where each R_j again represents only the rotation due to link j. The columns of the R matrix product are the vectors ${}^0\vec{n}_{i-1}$, ${}^0\vec{o}_{i-1}$, and ${}^0\vec{a}_{i-1}$.

$$R_1 R_2 \ldots R_{i-1} \frac{d}{dt}\,{}^{i-1}\vec{d} = R_1 R_2 \ldots R_{i-1} \left[\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \frac{d}{dt} d_i + {}^{i-1}\begin{bmatrix} -p_y \\ p_x \\ 0 \end{bmatrix}_5 \frac{d}{dt}\theta_i \right] \tag{6.26}$$

$$\frac{d}{dt}\,{}^0\vec{d} = {}^0\begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix}_{i-1} \left[\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \frac{d}{dt} d_i + {}^{i-1}\begin{bmatrix} -p_y \\ p_x \\ 0 \end{bmatrix}_5 \frac{d}{dt}\theta_i \right] \tag{6.27}$$

$$\frac{d}{dt}\,{}^0\vec{d} = {}^0\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_{i-1} \frac{d}{dt} d_i + \left[-\,{}^0\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}_{i-1} {}^{i-1}p_{y_5} + {}^0\begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix}_{i-1} {}^{i-1}p_{x_5} \right] \frac{d}{dt}\theta_i \tag{6.28}$$

For a prismatic (sliding) joint, $\frac{d}{dt}\theta_i$ is zero and the second term in the above equation drops out. For a revolute joint, $\frac{d}{dt}d_i$ is zero and the first term drops out. This second case applies to each of the Armatron manipulator joints as they are all revolute; the prismatic case thus need not be considered further. Note that in the following equation for the translation rate due to one manipulator joint, the rate is more properly labeled to reflect its dependence on joint i.

$$\frac{d}{dt}\,{}^{0}\vec{d}_i = \left[-\,{}^{0}\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}_{i-1} {}^{i-1}p_{y_5} + {}^{0}\begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix}_{i-1} {}^{i-1}p_{x_5} \right] \frac{d}{dt}\theta_i \tag{6.29}$$

b. <u>Rotational Velocities</u>. The rotational velocities about the x-, y-, and z-axes which lie in the directions of the base coordinate frame axes are obtained by referring again to the transformations of link i. The rotations involved in link i are the α_i rotation about the x_{i-1} axis and the θ_i rotation about the z_{i-1}-axis. The α rotations are of constant value and thus have no effect on rotational rates. The θ rotations are those of the manipulator's revolute joints. In the case of a prismatic joint (again, of which the Armatron manipulator has none), there is not variable rotation and hence the rotational rate is zero. In the case of the i^{th} revolute joint, the variable rotation is about the z_{i-1}-axis. Let the vector $\vec{\delta}_i$ represent the entire rotation due to joint i. Then the components of $\vec{\delta}_i$ may be denoted with respect to coordinate frame $i-1$ as follows prior to differentiation to achieve the desired rate.

$${}^{i-1}\vec{\delta}_i = \begin{bmatrix} 0 \\ 0 \\ \theta_i \end{bmatrix} \tag{6.30}$$

$$\frac{d}{dt}\,{}^{i-1}\vec{\delta}_i = \begin{bmatrix} 0 \\ 0 \\ \frac{d}{dt}\theta_i \end{bmatrix} \tag{6.31}$$

As was done for $\vec{d}$ in the linear velocity case, the rotational rate is obtained with respect to the directions of the base coordinate frame axes by premultiplication of both sides of the equation by $R_1R_2...R_{i-1}$. The directions of a vector with respect to frame i are thus obtained in terms of the base frame's directions.

$$R_1R_2...R_{i-1}\frac{d}{dt}\,{}^{i-1}\vec{\delta}_i = R_1R_2...R_{i-1}\begin{bmatrix} 0 \\ 0 \\ \frac{d}{dt}\theta_i \end{bmatrix} \tag{6.32}$$

$$\frac{d}{dt}\,{}^{0}\vec{\delta}_i = {}^{0}\begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix}_{i-1}\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}\frac{d}{dt}\theta_i \tag{6.33}$$

$$\frac{d}{dt}\,{}^{0}\vec{\delta}_i = {}^{0}\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_{i-1}\frac{d}{dt}\theta_i \tag{6.34}$$

c. The Manipulator Jacobian. Equations (6.29) and (6.34) are combined to form the relationship for the linear and rotational velocities of the manipulator end frame with respect to the base frame directions due to joint i.

$$\frac{d}{dt}\,{}^{0}\!\begin{bmatrix} d_x \\ d_y \\ d_z \\ \delta_x \\ \delta_y \\ \delta_z \end{bmatrix}_i = \begin{bmatrix} -\,{}^{0}\!\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}_{i-1} {}^{i-1}p_{y_5} + {}^{0}\!\begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix}_{i-1} {}^{i-1}p_{x_5} \\ {}^{0}\!\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_{i-1} \end{bmatrix} \frac{d}{dt}\theta_i \qquad (6.35)$$

The components of vector ${}^{0}\vec{d}_i$ in Equation (6.29) are referred to here generically as components of displacement vector $\vec{d}$. It should be noted at this point that the vector approach used here has derived the rotational velocity about the base coordinate frame; as such, the rotational velocity contributes to the translational velocity. This contribution is included in the translational velocity as the velocity was derived with respect to the base coordinate frame. Thus the rotational velocity may be thought of as reflecting the rotational rates about a coordinate frame identical in orientation to that of the base, but centered at the coordinate frame of the manipulator end.

The combined effects of the rate of each joint on the rates of the manipulator end frame are realized by summing the effects of the individual joints.

$$\frac{d}{dt}\,{}^{0}\!\begin{bmatrix} d_x \\ d_y \\ d_z \\ \delta_x \\ \delta_y \\ \delta_z \end{bmatrix} = \sum_{i=1}^{5} \begin{bmatrix} -\,{}^{0}\!\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}_{i-1} {}^{i-1}p_{y_5} + {}^{0}\!\begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix}_{i-1} {}^{i-1}p_{x_5} \\ {}^{0}\!\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_{i-1} \end{bmatrix} \frac{d}{dt}\theta_i \qquad (6.35)$$

If the rotational rate $\frac{d}{dt}\theta_i$ in Equation (6.36) is taken to be 1, then the translational and rotational rates of the left hand side become simply the elements of the right hand side column vector, reflecting the rates due to a joint rate of unity. The vector elements of the left hand side of the following equation are labeled with the subscript 1 to indicate this relationship.

$$\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ {}^0\delta_{z1} \end{bmatrix}_i = \begin{bmatrix} -{}^0\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}_{i-1} {}^{i-1}p_{y_5} + {}^0\begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix}_{i-1} {}^{i-1}p_{x_5} \\ {}^0\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_{i-1} \end{bmatrix} \quad (6.37)$$

Utilizing this nomenclature, Equation (6.36) may now be written in matrix form as

$$\frac{d}{dt}\,{}^0\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix} = \begin{bmatrix} d_{x1_1} & d_{x1_2} & d_{x1_3} & d_{x1_4} & d_{x1_5} \\ d_{y1_1} & d_{y1_2} & d_{y1_3} & d_{y1_4} & d_{y1_5} \\ d_{z1_1} & d_{z1_2} & d_{z1_3} & d_{z1_4} & d_{z1_5} \\ \delta_{x1_1} & \delta_{x1_2} & \delta_{x1_3} & \delta_{x1_4} & \delta_{x1_5} \\ \delta_{y1_1} & \delta_{y1_2} & \delta_{y1_3} & \delta_{y1_4} & \delta_{y1_5} \\ \delta_{z1_1} & \delta_{z1_2} & \delta_{z1_3} & \delta_{z1_4} & \delta_{z1_5} \end{bmatrix} \frac{d}{dt} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} \quad (6.38)$$

The 6x5 matrix of Equation (6.38)'s right hand side is termed the Jacobian matrix for the Armatron manipulator; it should also be remembered that this particular Jacobian resulted from a vector approach to the translational velocity. Another approach [Paul81c] employing matrix manipulations results in a completely different matrix and corresponding interpretation.

The Jacobian matrix is obtained one column at a time by employing Equation (6.37). Each element of the form ${}^0e_{i-1}$ is obtained from a corresponding matrix product $A_1A_2...A_{i-1}$. Recall that all of the matrix products of this form were generated in Chapter 4. Each element ${}^{i-1}e_5$ is obtained from a matching matrix product $A_iA_{i+1}...A_5$. Note that the complete matrix product need not be carried out as the formula to be invoked here employs only elements from the product's fourth column, p_x and p_y.

The required p_x and p_y expressions are obtained here for subsequent substitutions in Equation (6.37). For ${}^4p_{x5}$ and ${}^4p_{y5}$, A_5 is used from Equation (4.67).

$$A_5 = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.39}$$

$${}^4p_{x_5} = 0 \tag{6.40}$$

$${}^4p_{y_5} = 0 \tag{6.41}$$

Equation (4.65) provides A_4 for multiplication with A_5, in turn yielding ${}^3p_{x5}$ and ${}^3p_{y5}$. As noted, the first three columns are not needed and thus not generated.

$$A_4A_5 = \begin{bmatrix} c_4 & 0 & s_4 & 0 \\ s_4 & 0 & -c_4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.42}$$

$$A_4A_5 = \begin{bmatrix} - & - & - & d_5s_4 \\ - & - & - & -d_5c_4 \\ - & - & - & 0 \\ - & - & - & 1 \end{bmatrix} \tag{6.43}$$

$${}^3p_{x_5} = d_5s_4 \tag{6.44}$$

$${}^3p_{y_5} = -d_5c_4 \tag{6.45}$$

Matrix A_3 from Equation (4.63) is used to obtain ${}^2p_{x5}$ and ${}^2p_{y5}$.

$$A_3A_4A_5 = \begin{bmatrix} c_3 & 0 & s_3 & a_3c_3 \\ s_3 & 0 & -c_3 & a_3s_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} - & - & - & d_5s_4 \\ - & - & - & -d_5c_4 \\ - & - & - & 0 \\ - & - & - & 1 \end{bmatrix} \tag{6.46}$$

$$A_3A_4A_5 = \begin{bmatrix} - & - & - & d_5c_3s_4 + a_3c_3 \\ - & - & - & d_5s_3s_4 + a_3s_3 \\ - & - & - & -d_5c_4 \\ - & - & - & 1 \end{bmatrix} \tag{6.47}$$

$${}^2p_{x_5} = (d_5s_4 + a_3)c_3 \tag{6.48}$$

$${}^2p_{y_5} = (d_5s_4 + a_3)s_3 \tag{6.49}$$

Next, A_2 is used from Equation (4.61) in producing ${}^1p_{x5}$ and ${}^1p_{y5}$.

$$A_2A_3A_4A_5 = \begin{bmatrix} c_2 & 0 & -s_2 & a_2c_2 \\ s_2 & 0 & c_2 & a_2s_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} - & - & - & (d_5s_4 + a_3)c_3 \\ - & - & - & (d_5s_4 + a_3)s_3 \\ - & - & - & -d_5c_4 \\ - & - & - & 1 \end{bmatrix} \tag{6.50}$$

$$A_2A_3A_4A_5 = \begin{bmatrix} - & - & - & (d_5s_4 + a_3)c_2c_3 + d_5s_2c_4 + a_2c_2 \\ - & - & - & (d_5s_4 + a_3)s_2c_3 - d_5c_2c_4 + a_2s_2 \\ - & - & - & -(d_5s_4 + a_3)s_3 \\ - & - & - & 1 \end{bmatrix} \tag{6.51}$$

$${}^1p_{x_5} = ((d_5s_4 + a_3)c_3 + a_2)c_2 + d_5s_2c_4 \tag{6.52}$$

$${}^1p_{y_5} = ((d_5s_4 + a_3)c_3 + a_2)s_2 - d_5c_2c_4 \tag{6.53}$$

Finally, ${}^0p_{x_5}$ and ${}^0p_{y_5}$ are obtained from the use of A_1 in Equation (4.59).

$$A_1A_2A_3A_4A_5 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} - & - & - & ((d_5s_4 + a_3)c_3 + a_2)c_2 + d_5s_2c_4 \\ - & - & - & ((d_5s_4 + a_3)c_3 + a_2)s_2 - d_5c_2c_4 \\ - & - & - & -(d_5s_4 + a_3)s_3 \\ - & - & - & 1 \end{bmatrix} \tag{6.54}$$

$$A_1A_2A_3A_4A_5 = \begin{bmatrix} - & - & - & (((d_5s_4 + a_3)c_3 + a_2)c_2 + d_5s_2c_4)c_1 - (d_5s_4 + a_3)s_1s_3 \\ - & - & - & (((d_5s_4 + a_3)c_3 + a_2)c_2 + d_5s_2c_4)s_1 + (d_5s_4 + a_3)c_1s_3 \\ - & - & - & ((d_5s_4 + a_3)c_3 + a_2)s_2 - d_5c_2c_4 \\ - & - & - & 1 \end{bmatrix} \tag{6.55}$$

$${}^0p_{x_5} = (((d_5s_4 + a_3)c_3 + a_2)c_2 + d_5s_2c_4)c_1 - (d_5s_4 + a_3)s_1s_3 \tag{6.56}$$

$$ {}^{0}p_{y_5} = (((d_5 s_4 + a_3)c_3 + a_2)c_2 + d_5 s_2 c_4)s_1 + (d_5 s_4 + a_3)c_1 s_3 \tag{6.57}$$

Equation (6.37) is now employed along with the various matrix product results from Chapter 4 to obtain the columns of the Jacobian matrix. The rates for joint 1 require matrix product ${}^{0}A_0$, which is simply the identity matrix.

$$ {}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_1 = \begin{bmatrix} -{}^{0}\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}_0 {}^{0}p_{y_5} + {}^{0}\begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix}_0 {}^{0}p_{x_5} \\ {}^{0}\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_0 \end{bmatrix} \tag{6.58}$$

$$ {}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_1 = \begin{bmatrix} -\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} {}^{0}p_{y_5} + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} {}^{0}p_{x_5} \\ \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix} \tag{6.59}$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_1 = \begin{bmatrix} -((((d_5 s_4 + a_3)c_3 + a_2)c_2 + d_5 s_2 c_4)s_1 + (d_5 s_4 + a_3)c_1 s_3) \\ (((d_5 s_4 + a_3)c_3 + a_2)c_2 + d_5 s_2 c_4)c_1 - (d_5 s_4 + a_3)s_1 s_3 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{6.60}
$$

The rates for joint 2 utilize matrix product ${}^{0}A_1$, or just A_1, from Equation (4.59) and ${}^{1}p_{x_5}$ and ${}^{1}p_{y_5}$ from Equations (6.52) and (6.53), respectively.

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_2 = \begin{bmatrix} -\,{}^{0}\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}_1 {}^{1}p_{y_5} + {}^{0}\begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix}_1 {}^{1}p_{x_5} \\ {}^{0}\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_1 \end{bmatrix} \tag{6.61}
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_2 = \begin{bmatrix} -\begin{bmatrix} c_1 \\ s_1 \\ 0 \end{bmatrix} {}^{1}p_{y_5} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} {}^{1}p_{x_5} \\ \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix} \end{bmatrix} \tag{6.62}
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_2 = \begin{bmatrix} -(((d_5 s_4 + a_3)c_3 + a_2)s_2 - d_5 c_2 c_4)c_1 \\ -(((d_5 s_4 + a_3)c_3 + a_2)s_2 - d_5 c_2 c_4)s_1 \\ ((d_5 s_4 + a_3)c_3 + a_2)c_2 + d_5 s_2 c_4 \\ s_1 \\ -c_1 \\ 0 \end{bmatrix} \tag{6.63}
$$

Matrix product ${}^{0}A_2 = A_1A_2$ from Equation (4.70) and ${}^{2}p_{x5}$ and ${}^{2}p_{y5}$ from Equations (6.48) and (6.49), respectively, are employed to obtain the rates for joint 3.

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_3 = \begin{bmatrix} -{}^{0}\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}_2 {}^{2}p_{y_5} + {}^{0}\begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix}_2 {}^{2}p_{x_5} \\ {}^{0}\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_2 \end{bmatrix} \tag{6.64}
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_3 = \begin{bmatrix} -\begin{bmatrix} c_1 c_2 \\ s_1 c_2 \\ s_2 \end{bmatrix}(d_5 s_4 + a_3)s_3 + \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix}(d_5 s_4 + a_3)c_3 \\ \begin{bmatrix} -c_1 s_2 \\ -s_1 s_2 \\ c_2 \end{bmatrix} \end{bmatrix} \tag{6.65}
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_3 = \begin{bmatrix} -(d_5s_4 + a_3)(c_1c_2s_3 + s_1c_3) \\ -(d_5s_4 + a_3)(s_1c_2s_3 - c_1c_3) \\ -(d_5s_4 + a_3)s_2s_3 \\ -c_1s_2 \\ -s_1s_2 \\ c_2 \end{bmatrix} \qquad (6.66)
$$

Next, the rates for joint 4 require matrix product ${}^{0}A_3 = A_1A_2A_3$ from Equation (4.72) and ${}^{3}p_{x_5}$ and ${}^{3}p_{y_5}$ from Equations (6.44) and (6.45), respectively.

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_4 = \begin{bmatrix} -{}^{0}\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}_3 {}^{3}p_{y_5} + {}^{0}\begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix}_3 {}^{3}p_{x_5} \\ {}^{0}\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_3 \end{bmatrix} \qquad (6.67)
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_4 = \begin{bmatrix} -\begin{bmatrix} c_1c_2c_3 - s_1s_3 \\ s_1c_2c_3 + c_1s_3 \\ s_2c_3 \end{bmatrix}(-d_5c_4) + \begin{bmatrix} -c_1s_2 \\ -s_1s_2 \\ c_2 \end{bmatrix}(d_5s_4) \\ \begin{bmatrix} c_1c_2s_3 + s_1c_3 \\ s_1c_2s_3 - c_1c_3 \\ s_2s_3 \end{bmatrix} \end{bmatrix} \qquad (6.68)
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_4 = \begin{bmatrix} d_5((c_1c_2c_3 - s_1s_3)c_4 - c_1s_2s_4) \\ d_5((s_1c_2c_3 + c_1s_3)c_4 - s_1s_2s_4) \\ d_5(s_2c_3c_4 + c_2s_4) \\ c_1c_2s_3 + s_1c_3 \\ s_1c_2s_3 - c_1c_3 \\ s_2s_3 \end{bmatrix} \tag{6.69}
$$

Lastly, the matrix product ${}^{0}A_4 = A_1A_2A_3A_4$ from Equation (4.74) and ${}^{4}p_{x_5}$ and ${}^{4}p_{y_5}$ from Equations (6.40) and (6.41), respectively, are employed to obtain the rates for joint 5.

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_5 = \begin{bmatrix} -\,{}^{0}\begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix}_4 {}^{4}p_{y_5} + {}^{0}\begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix}_4 {}^{4}p_{x_5} \\ {}^{0}\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}_4 \end{bmatrix} \tag{6.70}
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_5 = \begin{bmatrix} -\begin{bmatrix} (c_1c_2c_3 - s_1s_3)c_4 - c_1s_2s_4 \\ (s_1c_2c_3 + c_1s_3)c_4 - s_1s_2s_4 \\ s_2c_3c_4 + c_2s_4 \end{bmatrix}(0) + \begin{bmatrix} c_1c_2s_3 + s_1c_3 \\ s_1c_2s_3 - c_1c_3 \\ s_2s_3 \end{bmatrix}(0) \\ \begin{bmatrix} (c_1c_2c_3 - s_1s_3)s_4 + c_1s_2c_4 \\ (s_1c_2c_3 + c_1s_3)s_4 + s_1s_2c_4 \\ s_2c_3s_4 - c_2c_4 \end{bmatrix} \end{bmatrix} \tag{6.71}
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_5 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ (c_1c_2c_3 - s_1s_3)s_4 + c_1s_2c_4 \\ (s_1c_2c_3 + c_1s_3)s_4 + s_1s_2c_4 \\ s_2c_3s_4 - c_2c_4 \end{bmatrix} \tag{6.72}
$$

The column formulas in Equations (6.60), (6.63), (6.66), (6.69), and (6.72) are then combined to form the manipulator Jacobian for the vector approach.

2. Numerical Example. The joint control variable values to be used here as those for the examples of the preceding chapters.

$$
\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} = \begin{bmatrix} -115^\circ \\ 25^\circ \\ 50^\circ \\ 65^\circ \\ -35^\circ \end{bmatrix} \tag{6.73}
$$

Column 1 of the Jacobian is obtained from Equation (6.60).

$$^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_{1} = \begin{bmatrix} \begin{pmatrix} -((((100\sin 65^{\circ}+100)\cos 50^{\circ}+100)\cos 25^{\circ} \\ +100\sin 25^{\circ}\cos 65^{\circ})\sin(-115^{\circ}) \\ +(100\sin 65^{\circ}+100)\cos(-115^{\circ})\sin 50^{\circ}) \end{pmatrix} \\ \begin{pmatrix} (((100\sin 65^{\circ}+100)\cos 50^{\circ}+100)\cos 25^{\circ} \\ +100\sin 25^{\circ}\cos 65^{\circ})\cos(-115^{\circ}) \\ -(100\sin 65^{\circ}+100)\sin(-115^{\circ})\sin 50^{\circ} \end{pmatrix} \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{6.74}$$

$$^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_{1} = \begin{bmatrix} 260.692 \\ 39.566 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{6.75}$$

Column 2 of the Jacobian is obtained from Equation (6.60).

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_2 = \begin{bmatrix} \begin{pmatrix} -(((100\sin 65^\circ + 100)\cos 50^\circ + 100)\sin 25^\circ \\ -100\cos 25^\circ \cos 65^\circ)\cos(-115^\circ) \end{pmatrix} \\ \begin{pmatrix} -(((100\sin 65^\circ + 100)\cos 50^\circ + 100)\sin 25^\circ \\ -100\cos 25^\circ \cos 65^\circ)\sin(-115^\circ) \end{pmatrix} \\ \begin{pmatrix} ((100\sin 65^\circ + 100)\cos 50^\circ + 100)\cos 25^\circ \\ +100\sin 25^\circ \cos 65^\circ \end{pmatrix} \\ \sin(-115^\circ) \\ -\cos(-115^\circ) \\ 0 \end{bmatrix} \tag{6.76}
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_2 = \begin{bmatrix} 23.559 \\ 50.522 \\ 219.546 \\ -0.906 \\ 0.423 \\ 0 \end{bmatrix} \tag{6.77}
$$

Equation (6.66) provides the formula for column 3 of the Jacobian.

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_3 = \begin{bmatrix} \begin{pmatrix} -(100\sin 65^\circ + 100)(\cos(-115^\circ)\cos 25^\circ \sin 50^\circ \\ +\sin(-115^\circ)\cos 50^\circ) \end{pmatrix} \\ \begin{pmatrix} -(100\sin 65^\circ + 100)(\sin(-115^\circ)\cos 25^\circ \sin 50^\circ \\ -\cos(-115^\circ)\cos 50^\circ) \end{pmatrix} \\ -(100\sin 65^\circ + 100)\sin 25^\circ \sin 50^\circ \\ -\cos(-115^\circ)\sin 25^\circ \\ -\sin(-115^\circ)\sin 25^\circ \\ \cos 25^\circ \end{bmatrix} \tag{6.78}
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_3 = \begin{bmatrix} 166.988 \\ 68.164 \\ -61.716 \\ 0.179 \\ 0.383 \\ 0.906 \end{bmatrix} \tag{6.79}
$$

Next, column 4 of the Jacobian is obtained from Equation (6.69).

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_4 = \begin{bmatrix} \begin{pmatrix} 100((\cos(-115^\circ)\cos 25^\circ \cos 50^\circ - \sin(-115^\circ)\sin 50^\circ)\cos 65^\circ \\ - \cos(-115^\circ)\sin 25^\circ \sin 65^\circ) \end{pmatrix} \\ \begin{pmatrix} 100((\sin(-115^\circ)\cos 25^\circ \cos 50^\circ + \cos(-115^\circ)\sin 50^\circ)\cos 65^\circ \\ - \sin(-115^\circ)\sin 25^\circ \sin 65^\circ) \end{pmatrix} \\ 100(\sin 25^\circ \cos 50^\circ \cos 65^\circ + \cos 25^\circ \sin 65^\circ) \\ \cos(-115^\circ)\cos 25^\circ \sin 50^\circ + \sin(-115^\circ)\cos 50^\circ \\ \sin(-115^\circ)\cos 25^\circ \sin 50^\circ - \cos(-115^\circ)\cos 50^\circ \\ \sin 25^\circ \sin 50^\circ \end{bmatrix} \tag{6.80}
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_4 = \begin{bmatrix} 35.123 \\ -1.282 \\ 93.620 \\ -0.876 \\ -0.358 \\ 0.324 \end{bmatrix} \tag{6.81}
$$

Lastly, Equation (6.72) provides the formula for column 5 of the Jacobian.

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_5 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \begin{pmatrix} (\cos(-115^\circ)\cos 25^\circ \cos 50^\circ - \sin(-115^\circ)\sin 50^\circ)\sin 65^\circ \\ + \cos(-115^\circ)\sin 25^\circ \cos 65^\circ \end{pmatrix} \\ \begin{pmatrix} (\sin(-115^\circ)\cos 25^\circ \cos 50^\circ + \cos(-115^\circ)\sin 50^\circ)\sin 65^\circ \\ + \sin(-115^\circ)\sin 25^\circ \cos 65^\circ \end{pmatrix} \\ \sin 25^\circ \cos 50^\circ \sin 65^\circ - \cos 25^\circ \cos 65^\circ \end{bmatrix} \tag{6.82}
$$

$$
{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix}_5 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.331 \\ -0.934 \\ -0.137 \end{bmatrix} \tag{6.83}
$$

The Jacobian matrix using the vector approach for the manipulator at the given positions of the joint variables is then

$$
\text{Jacobian} = \begin{bmatrix} 260.692 & 23.559 & 166.988 & 35.123 & 0 \\ 39.566 & 50.522 & 68.164 & -1.282 & 0 \\ 0 & 219.546 & -61.716 & 93.620 & 0 \\ 0 & -0.906 & 0.179 & -0.876 & 0.331 \\ 0 & 0.423 & 0.383 & -0.358 & -0.934 \\ 1 & 0 & 0.906 & 0.324 & -0.137 \end{bmatrix} \tag{6.84}
$$

The numerical example continues by employing the Jacobian as in Equation (6.38) to determine the translational and rotational rates corresponding to a particular set of joint variable rates.

$$\frac{d}{dt}\,{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix} = [\text{Jacobian}]\,\frac{d}{dt}\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} \tag{6.85}$$

The joint rates used here are selected arbitrarily.

$$\frac{d}{dt}\,{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix} = \begin{bmatrix} 260.692 & 23.559 & 166.988 & 35.123 & 0 \\ 39.566 & 50.522 & 68.164 & -1.282 & 0 \\ 0 & 219.546 & -61.716 & 93.620 & 0 \\ 0 & -0.906 & 0.179 & -0.876 & 0.331 \\ 0 & 0.423 & 0.383 & -0.358 & -0.934 \\ 1 & 0 & 0.906 & 0.324 & -0.137 \end{bmatrix}\begin{bmatrix} 0.1 \\ -0.15 \\ 1.0 \\ 0.2 \\ -0.1 \end{bmatrix} \tag{6.86}$$

$$\frac{d}{dt}\,{}^{0}\begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \\ \delta_{x1} \\ \delta_{y1} \\ \delta_{z1} \end{bmatrix} = \begin{bmatrix} 196.548 \\ 64.286 \\ -75.924 \\ 0.107 \\ 0.341 \\ 1.085 \end{bmatrix} \tag{6.87}$$

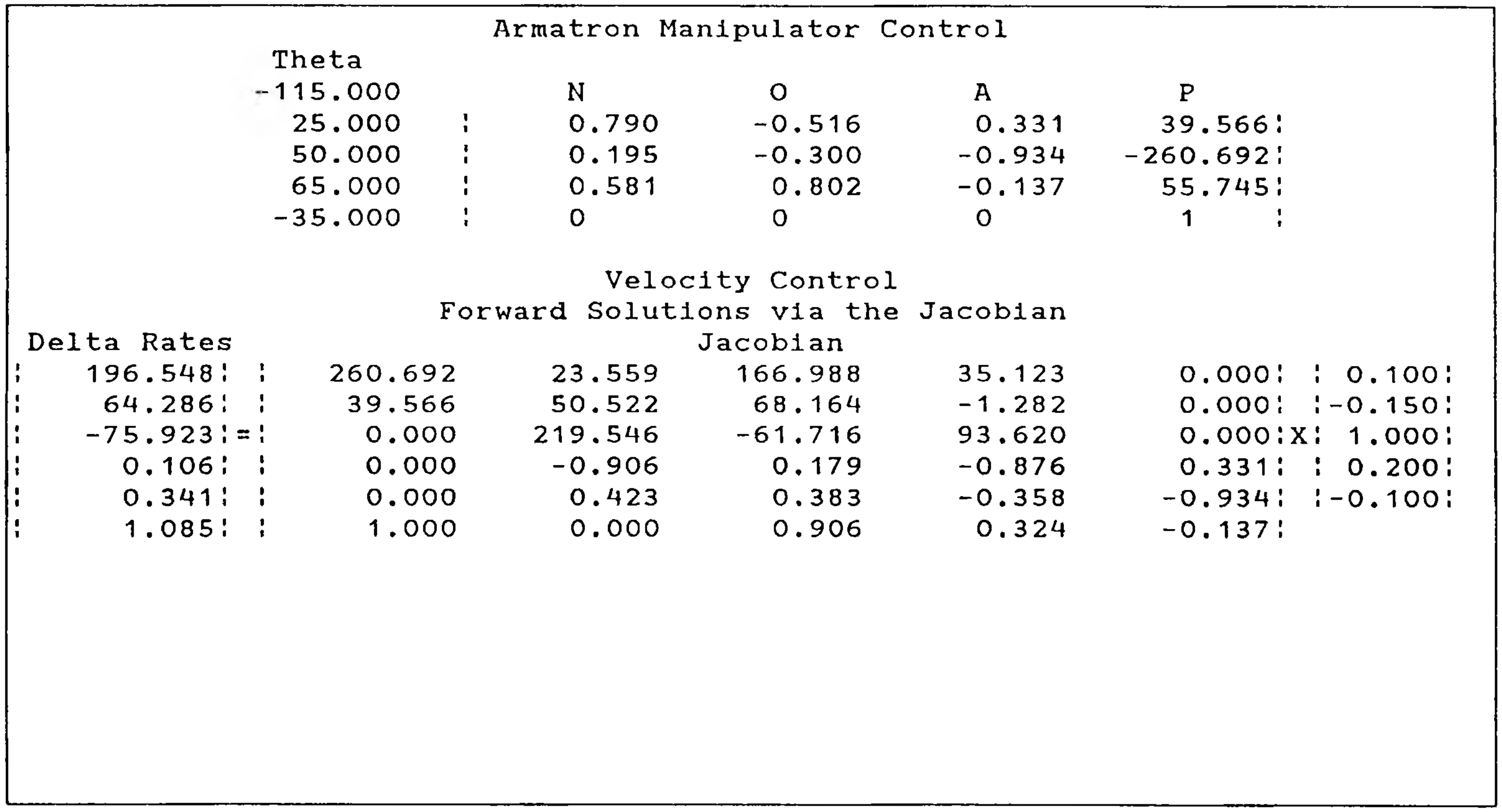
3. Program Control. The program procedure for the determination of translational and rotational rates begins by evaluating the sine and cosine of the

current joint control variables; the procedures then processes sets of delta theta rates. The iteration sequence begins by redrawing the display and refreshing the Jacobian, and then continues by prompting the user for a new set of joint rates. With these in hand, the corresponding translational and rotational rates are calculated and displayed. The iteration concludes by querying the user as to whether to continue in this mode. The body of the loop, given in procedure `for_sol_via_jac`, follows.

```
sin_cos (theta, s, c);
do
   {
   dsply_jacobian (&row, cols);
   calc_jacobian (s, c, jacobian, row, cols);
   get_delta_theta (dtheta, row, cols[6]);
   calc_list_rates (drate, dtheta, jacobian, row, cols[5]);
   query_ch = cont ("new Jacobian and/or theta rates");
   }
   while ( query_ch == 'Y' );
```

The logic of each of the procedures involved follows the derivation of equations in a straightforward manner and warrants no further explanation here.

4. Program Example. The program output for the set of joint values and joint rates specified in the numerical example of this section may be seen in Figure 6.2. The Jacobian will be seen to match precisely that calculated in Equation (6.86), while the translational and rotational rates pictured differ from those in Equation (6.87) by an amount small enough to attribute to precision.



```
                        Armatron Manipulator Control
            Theta
          -115.000           N          O          A          P
            25.000   |     0.790     -0.516      0.331     39.566|
            50.000   |     0.195     -0.300     -0.934   -260.692|
            65.000   |     0.581      0.802     -0.137     55.745|
           -35.000   |     0 0        0          0          1    |

                              Velocity Control
                      Forward Solutions via the Jacobian
 Delta Rates                              Jacobian
|   196.548| |   260.692     23.559    166.988     35.123      0.000| | 0.100|
|    64.286| |    39.566     50.522     68.164     -1.282      0.000| |-0.150|
|   -75.923|=|     0.000    219.546    -61.716     93.620      0.000|X| 1.000|
|     0.106| |     0.000     -0.906      0.179     -0.876      0.331| | 0.200|
|     0.341| |     0.000      0.423      0.383     -0.358     -0.934| |-0.100|
|     1.085| |     1.000      0.000      0.906      0.324     -0.137|
```

Figure 6.2. Forward Solutions via Jacobian Display

B. JOINT RATES VIA THE INVERSE JACOBIAN

The previous section obtained translational and rotational rates for the manipulator end coordinate frame in terms of the rotation rates of the Armatron joints. While this relationship is useful, its inverse is more so. For practical applications, the required rates of the manipulator end, particularly the translational, are generally known, and the joint rates must be determined. This section will derive relationships in this direction by inverting and solving numerically the differential equation from the first section of this chapter which provided end frame rates in terms of joint rates. In the next section, the equations derived in Chapter 5 will be differentiated to provide the joint rates directly.

The rate equation

$$\frac{d}{dt}\begin{bmatrix} d_x \\ d_y \\ d_z \\ \delta_x \\ \delta_y \\ \delta_z \end{bmatrix} = \begin{bmatrix} d_{x1_1} & d_{x1_2} & d_{x1_3} & d_{x1_4} & d_{x1_5} \\ d_{y1_1} & d_{y1_2} & d_{y1_3} & d_{y1_4} & d_{y1_5} \\ d_{z1_1} & d_{z1_2} & d_{z1_3} & d_{z1_4} & d_{z1_5} \\ \delta_{x1_1} & \delta_{x1_2} & \delta_{x1_3} & \delta_{x1_4} & \delta_{x1_5} \\ \delta_{y1_1} & \delta_{y1_2} & \delta_{y1_3} & \delta_{y1_4} & \delta_{y1_5} \\ \delta_{z1_1} & \delta_{z1_2} & \delta_{z1_3} & \delta_{z1_4} & \delta_{z1_5} \end{bmatrix} \frac{d}{dt}\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} \qquad (6.88)$$

is to be solved by numerical means. Unfortunately, while there are six translational and rotational rates to be specified, the Armatron manipulator consists of five, not six, degrees of freedom. Thus the Armatron Jacobian is a 6x5 matrix and has no inverse. There are however techniques which can be used to obtain solutions in such cases. The number of the rates of the left hand side of Equation (6.88) that need to be specified determines which of two methods to use. Each method determines a pseudo-inverse of the Jacobian matrix which allows Equation (6.88) to be solved.

1. The Over-Determined Case. First, assume it is desired to determine the joint rates needed to result in all six of the translational and rotational rates; this is termed the over-determined case, in that there are more equations to be satisfied than there are unknown variables.

a. Derivation of Equations. Equation (6.88) is restated in an abbreviated fashion to facilitate its manipulation.

$$d = Jq \tag{6.89}$$

For any set of joint rates q' chosen as a solution for this equation, the error associated with it is defined as

$$e = Jq' - d \tag{6.90}$$

where e is a 6x1 vector; each of the components of e would be zero for a perfect solution. A more workable form for the error associated with a solution is obtained by summing the squares of each element of e. This measure of error is termed e^2 and may be viewed as either the dot product of e with itself or e's transpose multiplied by e itself.

$$e^2 = e \cdot e \tag{6.91}$$

$$e^2 = e^T e \tag{6.92}$$

Substituting from Equation (6.90),

$$e^2 = (Jq' - d)^T (Jq' - d) \tag{6.93}$$

$$e^2 = (Jq')^T (Jq') - (Jq')^T d - d^T Jq' - d^T d \tag{6.94}$$

$$e^2 = q'^T J^T J q' - q'^T J^T d - d^T J q' - d^T d \tag{6.95}$$

Since each of the matrix products in Equation (6.95) results in a 1x1 matrix, which is a simple scalar value, any term of the equation may be transposed without affecting the result. The third term in the right hand side is thus transposed so that terms may be combined.

$$e^2 = q'^T J^T J q' - q'^T J^T d - (d^T J q')^T - d^T d \tag{6.96}$$

$$e^2 = q'^T J^T J q' - q'^T J^T d - q'^T J^T d - d^T d \tag{6.97}$$

$$e^2 = q'^T J^T J q' - 2q'^T J^T d - d^T d \tag{6.98}$$

The function defining e^2 in Equation (6.98) will take on a minimum value when its derivative is zero. Since e^2 is a function of all five of the joint rates in q', Equation (6.98) must be differentiated with respect to each of the variables separately. This will result in a set of five equations in five unknowns, which may then be solved. To see the form the solution for this situation takes, an example shall be demonstrated for a small case. Assume that J is a 3x2 matrix, q is a 2 element vector, and d is a 3 element vector. Equation (6.98) then becomes

$$\begin{aligned} e^2 = [q_1 \quad q_2] \begin{bmatrix} J_{11} & J_{21} & J_{31} \\ J_{12} & J_{22} & J_{32} \end{bmatrix} \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \\ J_{31} & J_{32} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \end{bmatrix} \\ - 2[q_1 \quad q_2] \begin{bmatrix} J_{11} & J_{21} & J_{31} \\ J_{12} & J_{22} & J_{32} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} - [d_1 \quad d_2 \quad d_3] \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} \end{aligned} \tag{6.99}$$

$$e^2 = [q_1 \quad q_2]\begin{bmatrix} J_{11}^2 + J_{21}^2 + J_{31}^2 & J_{11}J_{12} + J_{21}J_{22} + J_{31}J_{32} \\ J_{12}J_{11} + J_{22}J_{21} + J_{32}J_{31} & J_{12}^2 + J_{22}^2 + J_{32}^2 \end{bmatrix}\begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$$
$$- 2[J_{11}q_1 + J_{12}q_2 \quad J_{21}q_1 + J_{22}q_2 \quad J_{31}q_1 + J_{32}q_2]\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} - (d_1^2 + d_2^2 + d_3^2) \tag{6.100}$$

$$e^2 = [q_1 \quad q_2]\begin{bmatrix} (J_{11}^2 + J_{21}^2 + J_{31}^2)q_1 + (J_{11}J_{12} + J_{21}J_{22} + J_{31}J_{32})q_2 \\ (J_{12}J_{11} + J_{22}J_{21} + J_{32}J_{31})q_1 + (J_{12}^2 + J_{22}^2 + J_{32}^2)q_2 \end{bmatrix}$$
$$- 2((J_{11}q_1 + J_{12}q_2)d_1 + (J_{21}q_1 + J_{22}q_2)d_2 + (J_{31}q_1 + J_{32}q_2)d_3) - (d_1^2 + d_2^2 + d_3^2) \tag{6.101}$$

$$\begin{aligned} e^2 = {} & ((J_{11}^2 + J_{21}^2 + J_{31}^2)q_1^2 + (J_{12}J_{11} + J_{22}J_{21} + J_{32}J_{31})q_2q_1 \\ & + (J_{11}J_{12} + J_{21}J_{22} + J_{31}J_{32})q_1q_2 + (J_{12}^2 + J_{22}^2 + J_{32}^2)q_2^2) \\ & - 2(J_{11}d_1q_1 + J_{12}d_1q_2 + J_{21}d_2q_1 + J_{22}d_2q_2 + J_{31}d_3q_1 + J_{32}d_3q_2) - (d_1^2 + d_2^2 + d_3^2) \end{aligned} \tag{6.102}$$

$$\begin{aligned} e^2 = {} & (J_{11}^2 + J_{21}^2 + J_{31}^2)q_1^2 + 2(J_{11}J_{12} + J_{21}J_{22} + J_{31}J_{32})q_1q_2 + (J_{12}^2 + J_{22}^2 + J_{32}^2)q_2^2 \\ & - 2(J_{11}d_1 + J_{21}d_2 + J_{31}d_3)q_1 - 2(J_{12}d_1 + J_{22}d_2 + J_{32}d_3)q_2 - (d_1^2 + d_2^2 + d_3^2) \end{aligned} \tag{6.103}$$

The error is then differentiated with respect to the variables q_1 and q_2, and the derivatives are set equal to zero.

$$\begin{aligned} \frac{\partial}{\partial q_1} e^2 = {} & 2(J_{11}^2 + J_{21}^2 + J_{31}^2)q_1 + 2(J_{11}J_{12} + J_{21}J_{22} + J_{31}J_{32})q_2 \\ & - 2(J_{11}d_1 + J_{21}d_2 + J_{31}d_3) \end{aligned} \tag{6.104}$$

$$\begin{aligned} \frac{\partial}{\partial q_2} e^2 = {} & 2(J_{11}J_{12} + J_{21}J_{22} + J_{31}J_{32})q_1 + 2(J_{12}^2 + J_{22}^2 + J_{32}^2)q_2 \\ & - 2(J_{12}d_1 + J_{22}d_2 + J_{32}d_3) \end{aligned} \tag{6.105}$$

$$\begin{aligned} & 2(J_{11}^2 + J_{21}^2 + J_{31}^2)q_1 + 2(J_{11}J_{12} + J_{21}J_{22} + J_{31}J_{32})q_2 \\ & - 2(J_{11}d_1 + J_{21}d_2 + J_{31}d_3) = 0 \end{aligned} \tag{6.106}$$

$$2(J_{11}J_{12} + J_{21}J_{22} + J_{31}J_{32})q_1 + 2(J_{12}^2 + J_{22}^2 + J_{32}^2)q_2 - 2(J_{12}d_1 + J_{22}d_2 + J_{32}d_3) = 0 \tag{6.107}$$

Equations (6.106) and (6.107) are then combined by factoring them in terms of the matrices of Equation (6.99).

$$2\begin{bmatrix} J_{11}^2 + J_{21}^2 + J_{32}^2 & J_{11}J_{12} + J_{21}J_{22} + J_{31}J_{32} \\ J_{11}J_{12} + J_{21}J_{22} + J_{31}J_{32} & J_{12}^2 + J_{22}^2 + J_{32}^2 \end{bmatrix}\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} - 2\begin{bmatrix} J_{11} & J_{21} & J_{31} \\ J_{12} & J_{22} & J_{32} \end{bmatrix}\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix} = 0 \tag{6.108}$$

$$\begin{bmatrix} J_{11} & J_{21} & J_{31} \\ J_{12} & J_{22} & J_{32} \end{bmatrix}\begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \\ J_{31} & J_{32} \end{bmatrix} q' - J^T d = 0 \tag{6.109}$$

$$J^T J q' = J^T d \tag{6.110}$$

$$q' = (J^T J)^{-1} J^T d \tag{6.111}$$

Equation (6.111) is termed the least squares solution to a system of equations of the form of Equation (6.89). Matrix product $(J^T J)^{-1} J^T$ is referred to as the pseudo inverse Jacobian for the over-determined case. See [Stra80] for a geometric derivation of this same result.

Two points need to be made here concerning this solution. First, for the implementation of this solution, Ranky and Ho [Rank85] recommend that the inverse of $J^T J$ not be determined for use in Equation (6.111). Instead, q' is obtained by solving the set of equations specified by Equation (6.110). This reduces the number of computations required for the solution. The other point that should be made is that the solution developed above is only an approximate solution. In all likelihood, there is no perfect solution for the over-determined case. As a robot manipulator is a precise mechanism, the solution developed here may not be sufficient to meet the stated requirements. The techniques of the next section may be implemented instead.

b. Numerical Example. The inverse Jacobian method for the over-determined case is demonstrated by obtaining the original joint rates for the translational and rotational rates produced by the forward application of the Jacobian in the first section of this chapter and stated in Equation (6.87).

$$\frac{d}{dt}\,{}^{0}\!\begin{bmatrix} d_x \\ d_y \\ d_z \\ \delta_x \\ \delta_y \\ \delta_z \end{bmatrix} = \begin{bmatrix} 196.548 \\ 64.286 \\ -75.924 \\ 0.107 \\ 0.341 \\ 1.085 \end{bmatrix} \tag{6.112}$$

The Jacobian for the matrix approach was stated in Equation (6.84).

$$J = \begin{bmatrix} 260.692 & 23.559 & 166.988 & 35.123 & 0 \\ 39.566 & 50.522 & 68.164 & -1.282 & 0 \\ 0 & 219.546 & -61.716 & 93.620 & 0 \\ 0 & -0.906 & 0.179 & -0.876 & 0.331 \\ 0 & 0.423 & 0.383 & -0.358 & -0.934 \\ 1 & 0 & 0.906 & 0.324 & -0.137 \end{bmatrix} \tag{6.113}$$

The first step is to obtain matrix product J^TJ.

$$J^TJ = \begin{bmatrix} 260.692 & 39.566 & 0 & 0 & 0 & 1 \\ 23.559 & 50.522 & 219.546 & -0.906 & 0.423 & 0 \\ 166.988 & 68.164 & -61.716 & 0.179 & 0.383 & 0.906 \\ 35.123 & -1.282 & 93.620 & -0.876 & -0.358 & 0.324 \\ 0 & 0 & 0 & 0.331 & -0.934 & -0.137 \end{bmatrix} \begin{bmatrix} 260.692 & 23.559 & 166.988 & 35.123 & 0 \\ 39.566 & 50.522 & 68.164 & -1.282 & 0 \\ 0 & 219.546 & -61.716 & 93.620 & 0 \\ 0 & -0.906 & 0.179 & -0.876 & 0.331 \\ 0 & 0.423 & 0.383 & -0.358 & -0.934 \\ 1 & 0 & 0.906 & 0.324 & -0.137 \end{bmatrix} \quad (6.114)$$

$$J^TJ = \begin{bmatrix} 69526.787 & 8140.596 & 46230.319 & 9105.886 & -0.137 \\ 8140.596 & 51308.945 & -6171.649 & 21317.232 & -0.695 \\ 46230.319 & -6171.649 & 36341.187 & -0.119 & -0.423 \\ 9105.886 & 21317.232 & -0.119 & 10000.974 & 0 \\ -0.137 & -0.695 & -0.423 & 0 & 1.001 \end{bmatrix} \quad (6.115)$$

The second step is to determine the matrix product J^Td.

$$J^Td = \begin{bmatrix} 260.692 & 39.566 & 0 & 0 & 0 & 1 \\ 23.559 & 50.522 & 219.546 & -0.906 & 0.423 & 0 \\ 166.988 & 68.164 & -61.716 & 0.179 & 0.383 & 0.906 \\ 35.123 & -1.282 & 93.620 & -0.876 & -0.358 & 0.324 \\ 0 & 0 & 0 & 0.331 & -0.934 & -0.137 \end{bmatrix} \begin{bmatrix} 196.548 \\ 64.286 \\ -75.924 \\ 0.107 \\ 0.341 \\ 1.085 \end{bmatrix} \quad (6.116)$$

$$J^T d = \begin{bmatrix} 53783.116 \\ -8790.432 \\ 41890.007 \\ -286.928 \\ -0.432 \end{bmatrix} \tag{6.117}$$

The set of equations specified by Equation (6.110) is then solved by row manipulations and backward substitution.

$$J^T J q' = J^T d \tag{6.118}$$

$$\begin{bmatrix} 69526.787 & 8140.596 & 46230.319 & 9105.886 & -0.137 \\ 8140.596 & 51308.945 & -6171.649 & 21317.232 & -0.695 \\ 46230.319 & -6171.649 & 36341.187 & -0.119 & -0.423 \\ 9105.886 & 21317.232 & -0.119 & 10000.974 & 0 \\ -0.137 & -0.695 & -0.423 & 0 & 1.001 \end{bmatrix} q' = \begin{bmatrix} 53783.116 \\ -8790.432 \\ 41890.007 \\ -286.928 \\ -0.432 \end{bmatrix} \tag{6.119}$$

$$\begin{bmatrix} 69526.787 & 8140.596 & 46230.319 & 9105.886 & -0.137 \\ 0 & 50355.797 & -11584.561 & 20251.063 & -0.679 \\ 0 & -11584.561 & 5601.346 & -6054.879 & -0.332 \\ 0 & 20251.063 & -6054.879 & 8808.381 & 0.018 \\ 0 & -0.679 & -0.332 & 0.018 & 1.001 \end{bmatrix} q' = \begin{bmatrix} 53783.116 \\ -15087.668 \\ 6128.098 \\ -7330.874 \\ -0.326 \end{bmatrix} \tag{6.120}$$

$$\begin{bmatrix} 69526.787 & 8140.596 & 46230.319 & 9105.886 & -0.137 \\ 0 & 50355.797 & -11584.561 & 20251.063 & -0.679 \\ 0 & 0 & 2936.269 & -1396.038 & -0.488 \\ 0 & 0 & -1396.038 & 664.223 & 0.291 \\ 0 & 0 & -0.488 & 0.291 & 1.001 \end{bmatrix} q' = \begin{bmatrix} 53783.116 \\ -15087.668 \\ 2657.117 \\ -1263.225 \\ -0.529 \end{bmatrix} \quad (6.121)$$

$$\begin{bmatrix} 69526.787 & 8140.596 & 46230.319 & 9105.886 & -0.137 \\ 0 & 50355.797 & -11584.561 & 20251.063 & -0.679 \\ 0 & 0 & 2936.269 & -1396.038 & -0.488 \\ 0 & 0 & 0 & 0.482 & 0.059 \\ 0 & 0 & 0 & 0.059 & 1.001 \end{bmatrix} q' = \begin{bmatrix} 53783.116 \\ -15087.668 \\ 2657.117 \\ 0.091 \\ -0.087 \end{bmatrix} \quad (6.122)$$

$$\begin{bmatrix} 69526.787 & 8140.596 & 46230.319 & 9105.886 & -0.137 \\ 0 & 50355.797 & -11584.561 & 20251.063 & -0.679 \\ 0 & 0 & 2936.269 & -1396.038 & -0.488 \\ 0 & 0 & 0 & 0.482 & 0.059 \\ 0 & 0 & 0 & 0 & 0.994 \end{bmatrix} q' = \begin{bmatrix} 53783.116 \\ -15087.668 \\ 2657.117 \\ 0.091 \\ -0.098 \end{bmatrix} \quad (6.123)$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} q' = \begin{bmatrix} 0.100 \\ -0.150 \\ 1.000 \\ 0.201 \\ -0.099 \end{bmatrix} \quad (6.124)$$

The original set of joint rates from Equation (6.86) was

$$\frac{d}{dt}\begin{bmatrix}\theta_1\\ \theta_2\\ \theta_3\\ \theta_4\\ \theta_5\end{bmatrix} = \begin{bmatrix}0.1\\ -0.15\\ 1.0\\ 0.2\\ -0.1\end{bmatrix} \tag{6.125}$$

The minor differences seen here are attributable to the lack of precision in the calculations. Note also that since a solution for the given Jacobian equation did in fact exist, it was obtained by the solution process. In general, exact results cannot be obtained for the over-determined case.

c. Program Control. The procedure for obtaining translational and rotational rates by way of an inverse of the Jacobian matrix is named `rev_sol_via_ij`. The body of the procedure follows:

```
sin_cos (theta, s, c);
do
   {
   dsply_rsvij_jacobian (&row, cols);
   calc_jacobian (s, c, jacobian, row, cols);
   t = get_required_rates (delta_trans_rot, jacobian, used,
                           jacobian_reduced, delta_tr_reduced,
                           row, cols);
   wait_then_erase (9);
   if ( (t > 0) & (t < 6) )
      {
      lcputs (10, 10, "Under-Determined Case");
      ic = under_determined_case (t, jacobian_reduced,
                                  delta_tr_reduced, delta_theta);
      }
   if ( t == 6 )
      {
      lcputs (10, 10, "Over-Determined Case");
      ic = over_determined_case (jacobian, delta_trans_rot,
                                 delta_theta);
      }
   if ( (t > 0) & (!ic) )
      list_input_output (delta_trans_rot, used, delta_theta);
   qc = cont ("different Jacobian and/or rates");
   }
   while ( qc == 'Y' );
```

As shown here, the iteration begins by refreshing the display and the Jacobian and then proceeds to query the user for the rates to be designated as command variables.

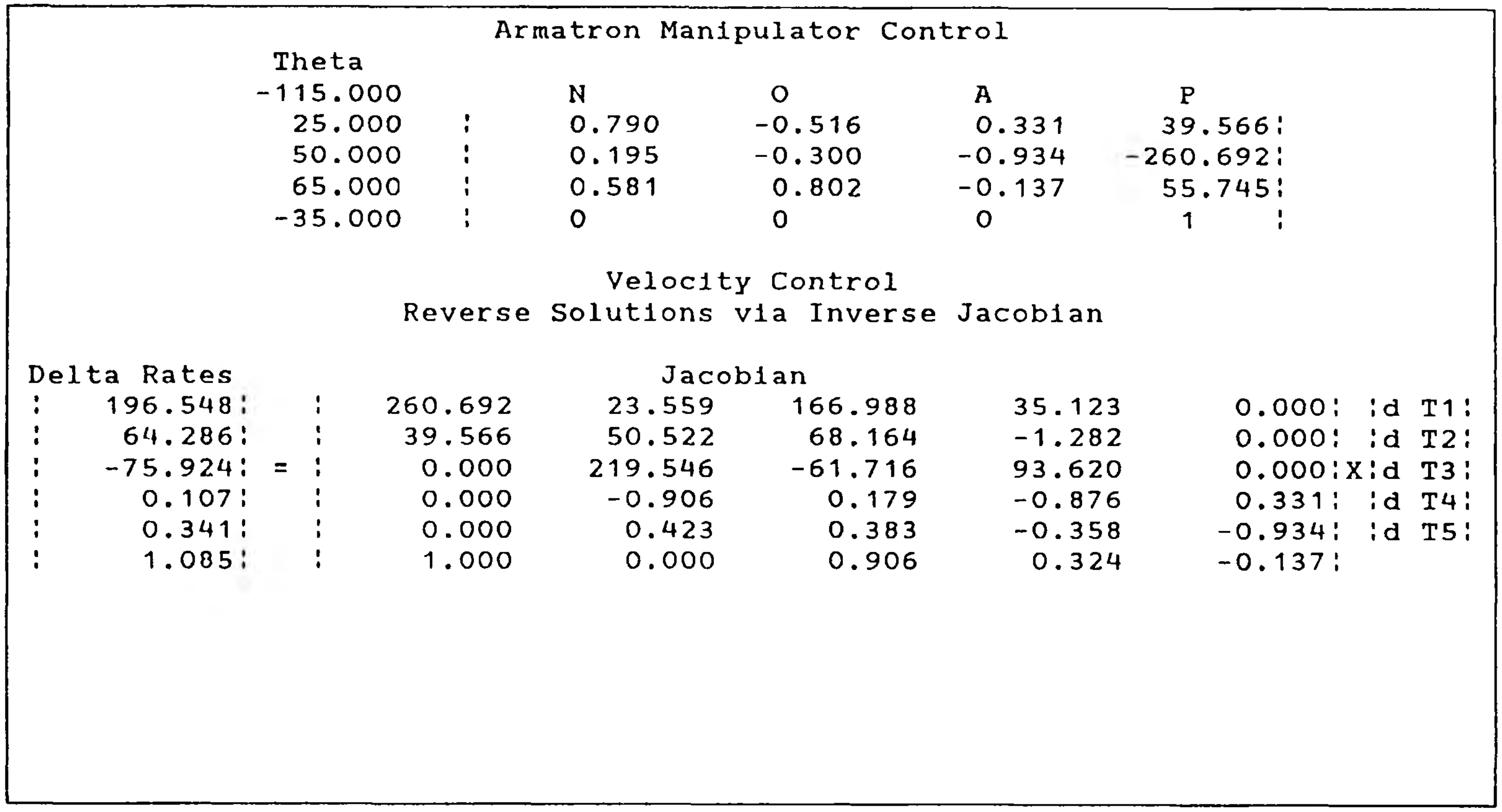
Procedure get_required_rates returns the number of rates designated for use; the procedure can then infer whether the case is over- or under-determined.

For the over-determined case, the program proceeds as described in the derivation of equations. Procedure over_determined_case receives control and executes the matrix operations required in the order required of the particular inverse Jacobian involved. The body of the procedure follows.

```
    subhead = "1.  M = (J Transpose) * J";
    matrix_by_matrix (6, jacobian, m, subhead);
    subhead = "2.  V = (J Transpose) * T/R Rates";
    matrix_by_vector (jacobian, 6, delta_trans_rot, v, subhead);
    subhead = "3.  Solve M * Theta Rates = V";
    inconsistent = solve_simul_eqns_myv (m, v, 5, delta_theta, subhead)
    return (inconsistent);
```

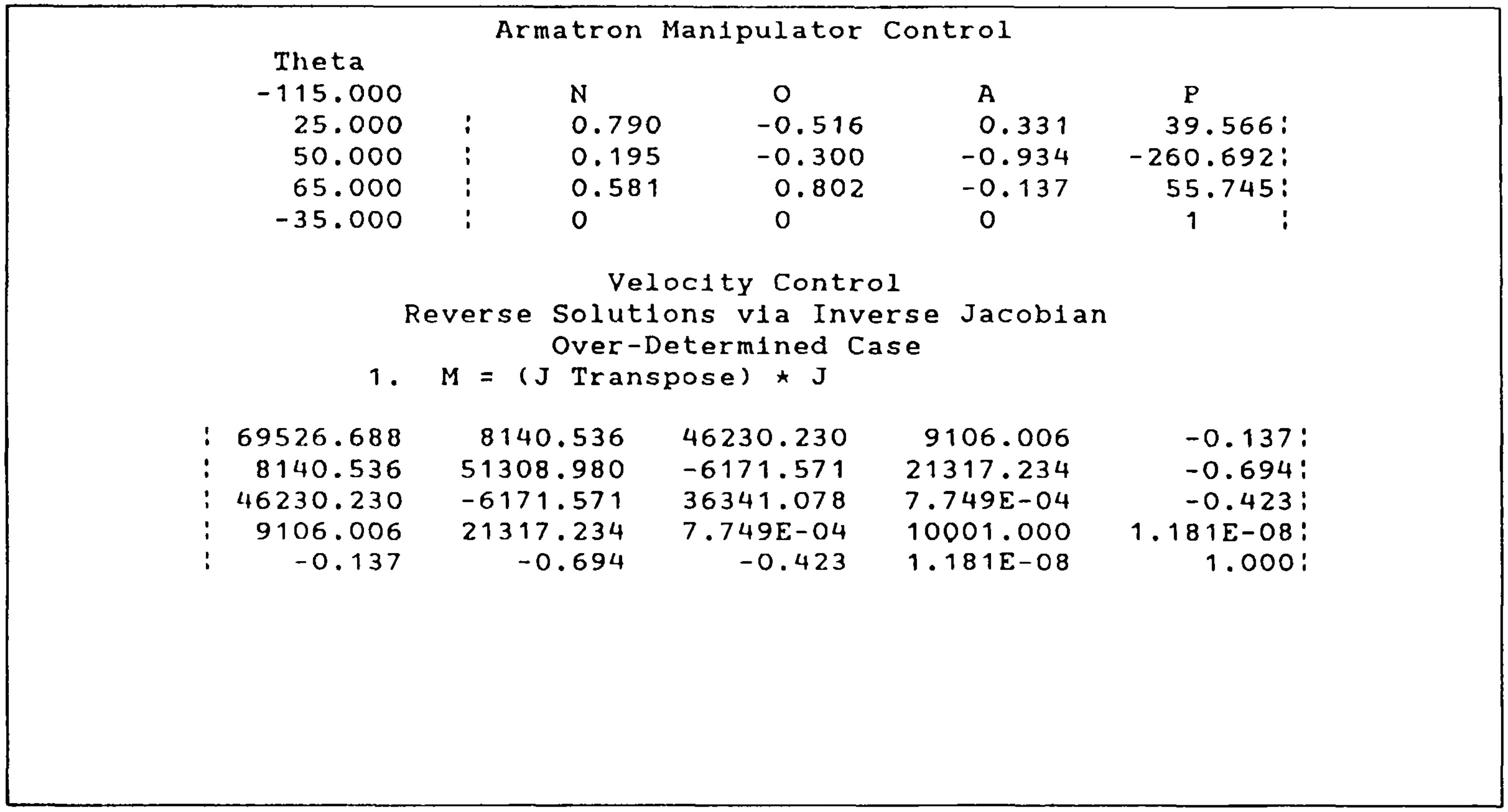
The procedure which solves a set of simultaneous equations returns a value of 0 if the equations specified were soluble.

d. Program Example. Figures 6.3 through 6.7 show the displays presented by the program for the solution of the over-determined case. The matrix multiplication displayed in Figure 6.4 differs only trivially from that calculated in Equation (6.115); the same is true of the values shown for the matrix multiplication of Figure 6.5 and the values in Equation (6.117). It is interesting to note that the final joint rates arrived at in Figure 6.6 are slightly further off the original set given by Equation (6.86) than are those calculated in Equation (6.124) in the numerical example.



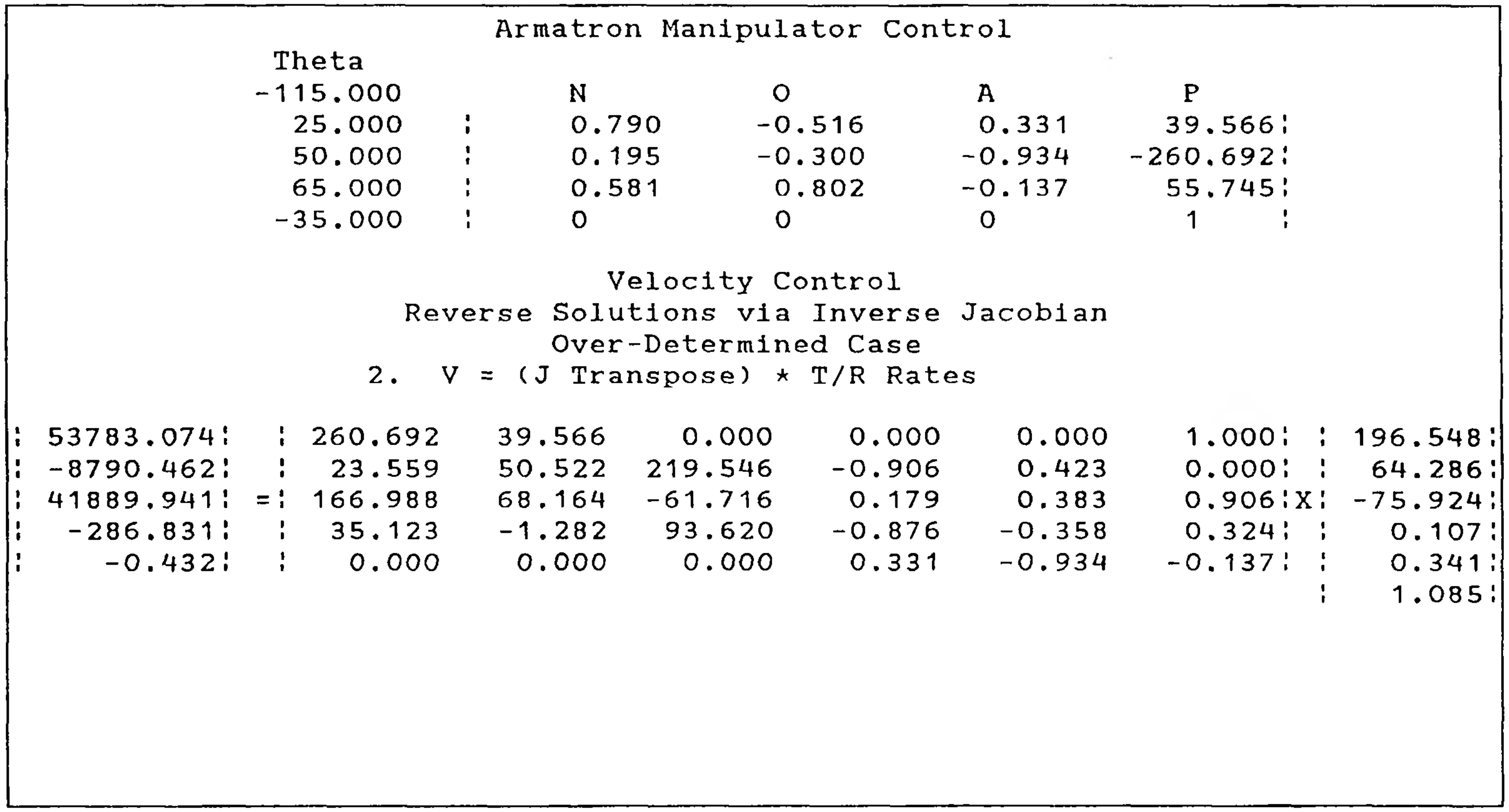
```
                    Armatron Manipulator Control
            Theta
           -115.000           N          O          A          P
             25.000   :     0.790     -0.516      0.331     39.566:
             50.000   :     0.195     -0.300     -0.934   -260.692:
             65.000   :     0.581      0.802     -0.137     55.745:
            -35.000   :     0 	       0          0          1     :

                          Velocity Control
                Reverse Solutions via Inverse Jacobian

Delta Rates                        Jacobian
:    196.548:    :   260.692     23.559    166.988     35.123      0.000: :d T1:
:     64.286:    :    39.566     50.522     68.164     -1.282      0.000: :d T2:
:    -75.924: =  :     0.000    219.546    -61.716     93.620      0.000:X:d T3:
:      0.107:    :     0.000     -0.906      0.179     -0.876      0.331: :d T4:
:      0.341:    :     0.000      0.423      0.383     -0.358     -0.934: :d T5:
:      1.085:    :     1.000      0.000      0.906      0.324     -0.137:
```

Figure 6.3. Inverse Jacobian, Over-Determined Example: Inputs



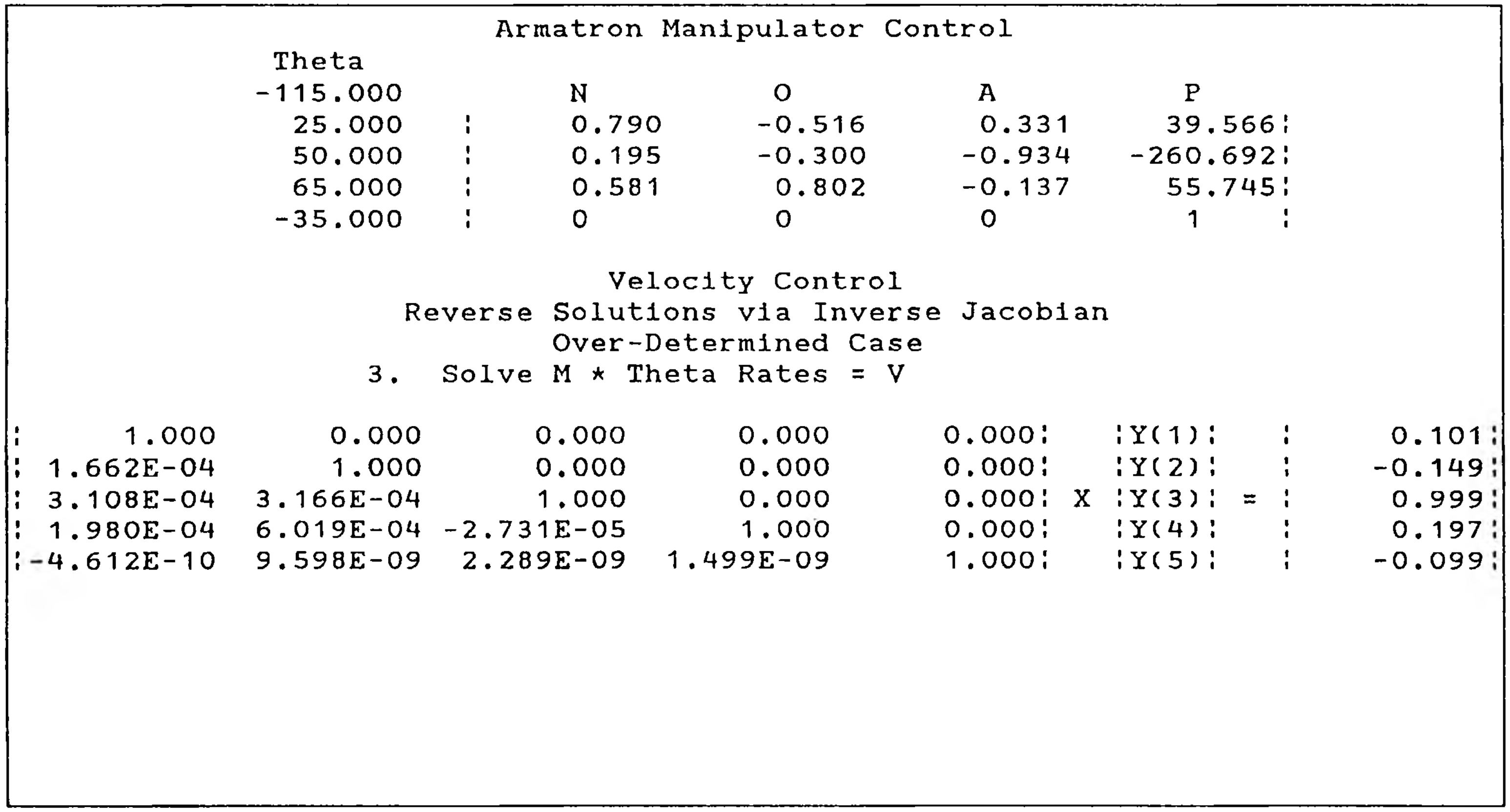
```
                    Armatron Manipulator Control
     Theta
    -115.000               N           O           A           P
      25.000   :        0.790      -0.516       0.331      39.566:
      50.000   :        0.195      -0.300      -0.934    -260.692:
      65.000   :        0.581       0.802      -0.137      55.745:
     -35.000   :        0             0           0          1    :

                              Velocity Control
                   Reverse Solutions via Inverse Jacobian
                            Over-Determined Case
               1.   M = (J Transpose) * J

 :  69526.688     8140.536    46230.230     9106.006        -0.137:
 :   8140.536    51308.980    -6171.571    21317.234        -0.694:
 :  46230.230    -6171.571    36341.078    7.749E-04        -0.423:
 :   9106.006    21317.234    7.749E-04    10001.000     1.181E-08:
 :     -0.137       -0.694       -0.423    1.181E-08         1.000:
```

Figure 6.4. Inverse Jacobian, Over-Determined Example: Step 1



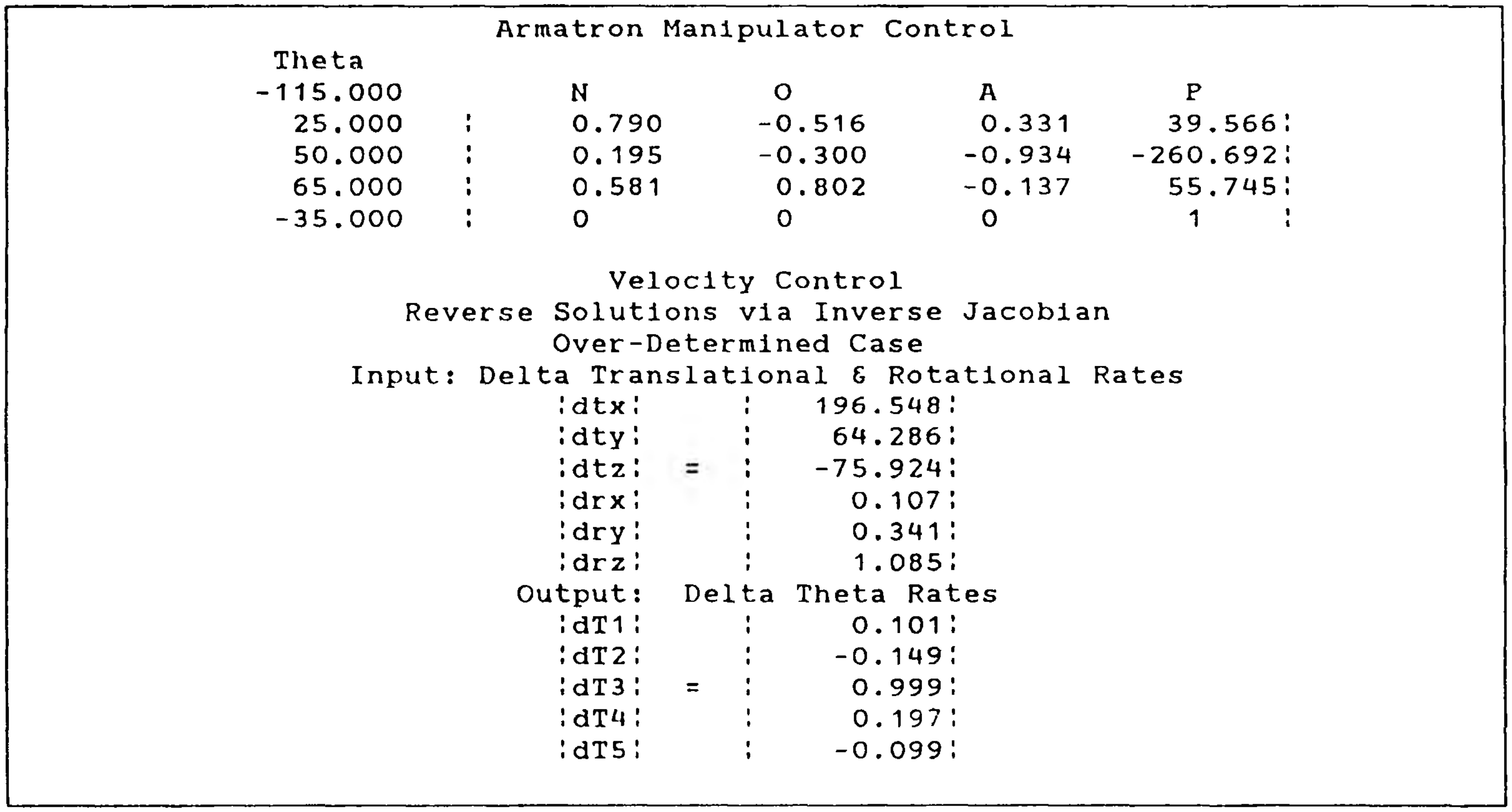
```
                         Armatron Manipulator Control
             Theta
            -115.000          N          O          A          P
              25.000   :     0.790     -0.516      0.331     39.566:
              50.000   :     0.195     -0.300     -0.934   -260.692:
              65.000   :     0.581      0.802     -0.137     55.745:
             -35.000   :     0              0          0          1     :

                               Velocity Control
                      Reverse Solutions via Inverse Jacobian
                              Over-Determined Case
                     2.   V = (J Transpose) * T/R Rates

: 53783.074:   : 260.692   39.566    0.000    0.000    0.000    1.000: : 196.548:
: -8790.462:   :  23.559   50.522  219.546   -0.906    0.423    0.000: :  64.286:
: 41889.941:  =: 166.988   68.164  -61.716    0.179    0.383    0.906:X: -75.924:
:  -286.831:   :  35.123   -1.282   93.620   -0.876   -0.358    0.324: :   0.107:
:    -0.432:   :   0.000    0.000    0.000    0.331   -0.934   -0.137: :   0.341:
                                                                       :   1.085:
```

Figure 6.5. Inverse Jacobian, Over-Determined Example: Step 2



```
                        Armatron Manipulator Control
              Theta
             -115.000              N           O           A           P
               25.000   :        0.790      -0.516       0.331      39.566:
               50.000   :        0.195      -0.300      -0.934    -260.692:
               65.000   :        0.581       0.802      -0.137      55.745:
              -35.000   :        0                0           0           1      :

                                  Velocity Control
                         Reverse Solutions via Inverse Jacobian
                                 Over-Determined Case
                          3.  Solve M * Theta Rates = V

:       1.000       0.000       0.000       0.000       0.000:   :Y(1):    :       0.101:
:   1.662E-04       1.000       0.000       0.000       0.000:   :Y(2):    :      -0.149:
:   3.108E-04   3.166E-04       1.000       0.000       0.000: X :Y(3): =  :       0.999:
:   1.980E-04   6.019E-04  -2.731E-05       1.000       0.000:   :Y(4):    :       0.197:
:  -4.612E-10   9.598E-09   2.289E-09   1.499E-09       1.000:   :Y(5):    :      -0.099:
```

Figure 6.6. Inverse Jacobian, Over-Determined Example: Step 3



```
                       Armatron Manipulator Control
      Theta
     -115.000          N          O          A          P
       25.000   :    0.790     -0.516      0.331     39.566:
       50.000   :    0.195     -0.300     -0.934   -260.692:
       65.000   :    0.581      0.802     -0.137     55.745:
      -35.000   :    0		   0          0          1    :

                            Velocity Control
                   Reverse Solutions via Inverse Jacobian
                          Over-Determined Case
               Input: Delta Translational & Rotational Rates
                          :dtx:      :   196.548:
                          :dty:      :    64.286:
                          :dtz:  =   :   -75.924:
                          :drx:      :     0.107:
                          :dry:      :     0.341:
                          :drz:      :     1.085:
                        Output:  Delta Theta Rates
                          :dT1:      :     0.101:
                          :dT2:      :    -0.149:
                          :dT3:  =   :     0.999:
                          :dT4:      :     0.197:
                          :dT5:      :    -0.099:
```

Figure 6.7. Inverse Jacobian, Over-Determined Example: Results

2. The Under-Determined Case. If it is not required that all six of the translational and rotational rates be met by a set of joint rates, then the under-determined case is in effect.

a. Derivation of Equations. With respect to Equation (6.88), only the desired translational and rotational rates are stated in the left hand vector, and the Jacobian is reduced to contain only the rows associated with them. For example,

$$\frac{d}{dt}\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} d_{x1_1} & d_{x1_2} & d_{x1_3} & d_{x1_4} & d_{x1_5} \\ d_{y1_1} & d_{y1_2} & d_{y1_3} & d_{y1_4} & d_{y1_5} \\ d_{z1_1} & d_{z1_2} & d_{z1_3} & d_{z1_4} & d_{z1_5} \end{bmatrix} \frac{d}{dt}\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} \tag{6.126}$$

$$d = Jq \tag{6.127}$$

There are now fewer equations than there are unknowns. Consequently, Equation (6.127) has an infinite number of solutions. However, there is one solution q' whose Euclidean norm, $q'\cdot q'$ or $q'^T q'$, is a minimum for the equation. This is the set of joint rates which are the smallest in magnitude producing the desired translational and/or rotational rates. This solution is found by applying the Lagrangian multiplier technique to Equation (6.127); see [Boul71] and [Rao71] for further detail on this technique. The function

$$\psi(q') = q'^T q' + \lambda(Jq' - d) \tag{6.128}$$

is to be minimized with

$$\lambda = [\lambda_1 \;\; \lambda_2 \;\; . \;\; . \;\; . \;\; \lambda_n] \tag{6.129}$$

where n is the number of translational and rotational rates specified. The function of Equation (6.128) will take on a minimum value when its derivative is zero. Like e^2 in the over-determined case, $\psi(q')$ is a function of all five of the joint rates in q', and Equation (6.128) must be differentiated with respect to each of the variables separately. Five equations in five unknowns result, and these equations may then be solved. As was noted for the over-determined case, the differentiation process need not be carried out explicitly. To see how the process works, a small example shall be examined as was done for the over-determined case. Assume that J is a 2x3 matrix, q is a 3 element vector, and d is a 2 element vector; λ is likewise a 2 element vector. Equation (6.128) then becomes

$$\psi(q') = \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} + \begin{bmatrix} \lambda_1 & \lambda_2 \end{bmatrix} \left[\begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} - \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \right] \tag{6.130}$$

$$\psi(q') = q_1^2 + q_2^2 + q_3^2 + \begin{bmatrix} \lambda_1 & \lambda_2 \end{bmatrix} \left[\begin{bmatrix} J_{11}q_1 + J_{12}q_2 + J_{13}q_3 \\ J_{21}q_1 + J_{22}q_2 + J_{23}q_3 \end{bmatrix} - \begin{bmatrix} d_1 \\ d_2 \end{bmatrix} \right]$$

$$\begin{aligned} \psi(q') = q_1^2 + q_2^2 + q_3^2 + \lambda_1(J_{11}q_1 + J_{12}q_2 + J_{13}q_3 - d_1) \\ + \lambda_2(J_{21}q_1 + J_{22}q_2 + J_{23}q_3 - d_2) \end{aligned} \tag{6.132}$$

Differentiation is then performed with respect to variables q_1, q_2, and q_3.

$$\frac{\partial}{\partial q_1} \psi(q') = 2q_1 + \lambda_1 J_{11} + \lambda_2 J_{21} \tag{6.133}$$

$$\frac{\partial}{\partial q_2} \psi(q') = 2q_2 + \lambda_1 J_{12} + \lambda_2 J_{22} \tag{6.134}$$

$$\frac{\partial}{\partial q_3} \psi(q') = 2q_3 + \lambda_1 J_{13} + \lambda_2 J_{23} \tag{6.135}$$

The derivatives of Equations (6.133), (6.134), and (6.135) may be combined into a single matrix equation upon equation with 0.

$$2\begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} + \begin{bmatrix} J_{11} & J_{21} \\ J_{12} & J_{22} \\ J_{13} & J_{23} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{6.136}$$

$$2q' + J^T\lambda^T = 0 \tag{6.137}$$

$$q' = -\frac{1}{2} J^T\lambda^T \tag{6.138}$$

This result for q' is then substituted into Equation (6.127) so that λ may be solved for in terms of J and d.

$$d = J(-\frac{1}{2} J^T\lambda^T) \tag{6.139}$$

$$-2d = JJ^T\lambda^T \tag{6.140}$$

$$\lambda^T = -2(JJ^T)^{-1}d \tag{6.141}$$

This result is then substituted back into Equation (6.138) to yield q' in terms of J and d.

$$q' = -\frac{1}{2} J^T(-2(JJ^T)^{-1}d) \tag{6.142}$$

$$q' = J^T(JJ^T)^{-1}d \tag{6.143}$$

Matrix product $J^T(JJ^T)^{-1}$ is referred to as the pseudo-inverse of the Jacobian for the under-determined case.

The number of calculations for the determination of q' may be reduced by a scheme similar to that used in the over-determined case. First, introduce a new matrix, r.

$$r = (JJ^T)^{-1}d \tag{6.144}$$

$$JJ^T r = d \tag{6.145}$$

Matrix r is determined by solving the specified system of equations; the inverse $(JJ^T)^{-1}$ need not be found. A substitution is then made into Equation (6.143).

$$q' = J^T r \tag{6.146}$$

b. Numerical Examples. The inverse Jacobian method for the under-determined case is first demonstrated by obtaining the optimal set of joint rates for only translational rates. The rates produced by the forward application of the Jacobian in the first section shall be used here as they were in the example of the over-determined case, as stated in Equation (6.112).

$$\frac{d}{dt}\,{}^{0}\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} = \begin{bmatrix} 196.548 \\ 64.286 \\ -75.924 \end{bmatrix} \tag{6.147}$$

The Jacobian determined in the first section and restated for the over-determined case in Equation (6.113) is reduced by eliminating the bottom three rows as they deal with the rotational rates.

$$J = \begin{bmatrix} 260.692 & 23.559 & 166.988 & 35.123 & 0 \\ 39.566 & 50.522 & 68.164 & -1.282 & 0 \\ 0 & 219.546 & -61.716 & 93.620 & 0 \end{bmatrix} \tag{6.148}$$

The first step is to obtain matrix product JJ^T.

$$JJ^T = \begin{bmatrix} 260.692 & 23.559 & 166.988 & 35.123 & 0 \\ 39.566 & 50.522 & 68.164 & -1.282 & 0 \\ 0 & 219.546 & -61.716 & 93.620 & 0 \end{bmatrix} \begin{bmatrix} 260.692 & 39.566 & 0 \\ 23.559 & 50.522 & 219.546 \\ 166.988 & 68.164 & -61.716 \\ 35.123 & -1.282 & 93.620 \\ 0 & 0 & 0 \end{bmatrix} \tag{6.149}$$

$$JJ^T = \begin{bmatrix} 97633.963 & 22842.330 & -1845.332 \\ 22842.330 & 8765.915 & 6765.073 \\ -1845.332 & 6765.073 & 60774.015 \end{bmatrix} \tag{6.150}$$

The second step is to determine matrix r. The set of equations specified by Equation (6.145) is solved by row manipulations and backward substitution.

$$JJ^T r = d \tag{6.151}$$

$$\begin{bmatrix} 97633.963 & 22842.330 & -1845.332 \\ 22842.330 & 8765.915 & 6765.073 \\ -1845.332 & 6765.073 & 60774.015 \end{bmatrix} r = \begin{bmatrix} 196.548 \\ 64.286 \\ -75.924 \end{bmatrix} \tag{6.152}$$

$$\begin{bmatrix} 97633.963 & 22842.330 & -1845.332 \\ 0 & 3421.750 & 7196.805 \\ 0 & 7196.805 & 60739.137 \end{bmatrix} r = \begin{bmatrix} 196.548 \\ 18.302 \\ -72.209 \end{bmatrix} \tag{6.153}$$

$$\begin{bmatrix} 97633.963 & 22842.330 & -1845.332 \\ 0 & 3421.750 & 7196.805 \\ 0 & 0 & 45602.437 \end{bmatrix} r = \begin{bmatrix} 196.548 \\ 18.302 \\ -110.703 \end{bmatrix} \tag{6.154}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} r = \begin{bmatrix} -4.800E-4 \\ 1.046E-2 \\ -2.428E-3 \end{bmatrix} \tag{6.155}$$

Finally, the resultant r is substituted into Equation (6.146).

$$q' = J^T r \tag{6.156}$$

$$q' = \begin{bmatrix} 260.692 & 39.566 & 0 \\ 23.559 & 50.522 & 219.546 \\ 166.988 & 68.164 & -61.716 \\ 35.123 & -1.282 & 93.620 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -4.800E-4 \\ 1.046E-2 \\ -2.428E-3 \end{bmatrix} \tag{6.157}$$

$$q' = \begin{bmatrix} 0.289 \\ -0.016 \\ 0.783 \\ -0.258 \\ 0 \end{bmatrix} \tag{6.158}$$

This solution set may be verified by using the Jacobian of Equation (6.113) in the forward fashion as in Equation (6.127).

$$d = Jq \tag{6.159}$$

$$d = \begin{bmatrix} 260.692 & 23.559 & 166.988 & 35.123 & 0 \\ 39.566 & 50.522 & 68.164 & -1.282 & 0 \\ 0 & -19.546 & -61.716 & 93.620 & 0 \\ 0 & -0.906 & 0.179 & -0.876 & 0.331 \\ 0 & 0.423 & 0.383 & -0.358 & -0.934 \\ 1 & 0 & 0.906 & 0.324 & -0.137 \end{bmatrix} \begin{bmatrix} 0.289 \\ -0.016 \\ 0.783 \\ -0.258 \\ 0 \end{bmatrix} \tag{6.160}$$

$$d = \begin{bmatrix} 196.653 \\ 64.329 \\ -75.990 \\ 0.381 \\ 0.385 \\ 0.915 \end{bmatrix} \tag{6.161}$$

Comparison with the original set of translational rates in Equation (6.147) shows no significant differences. However, the original rotational rates, stated in Equation (6.112), are not matched. The Euclidean norm of the newly obtained q' is less, however; it is in fact minimal.

$$|q| = < 0.1 \quad -0.15 \quad 1 \quad 0.2 \quad -0.1 > \cdot < 0.1 \quad -0.15 \quad 1 \quad 0.2 \quad -0.1 > \tag{6.162}$$

$$|q| = 1.083 \tag{6.163}$$

$$\begin{aligned} |q'| &= < 0.289 \quad -0.016 \quad 0.783 \quad -0.258 \quad 0 > \\ &\cdot < 0.289 \quad -0.016 \quad 0.783 \quad -0.258 \quad 0 > \end{aligned} \tag{6.164}$$

$$|q'| = 0.763 \tag{6.165}$$

Thus the desired translational rates are attained at overall slower rotational rates.

A second numerical example serves to illustrate the possibilities of the under-determined case. Instead of requiring the translational rates only, the y-translational as well as the x- and z-rotational rates are selected. The rates are again chosen from the solution obtained by forward application of the Jacobian in Equation (6.112).

$$\frac{d}{dt}\,{}^{0}\begin{bmatrix} d_y \\ \delta_x \\ \delta_z \end{bmatrix} = \begin{bmatrix} 64.286 \\ 0.107 \\ 1.085 \end{bmatrix} \tag{6.166}$$

The Jacobian matrix of Equation (6.113). is then reduced to the second, fourth, and sixth rows.

$$J = \begin{bmatrix} 39.566 & 50.522 & 68.164 & -1.282 & 0 \\ 0 & -0.906 & 0.179 & -0.876 & 0.331 \\ 1 & 0 & 0.906 & 0.324 & -0.137 \end{bmatrix} \tag{6.167}$$

The first step obtains matrix product JJ^T.

$$JJ^T = \begin{bmatrix} 39.566 & 50.522 & 68.164 & -1.282 & 0 \\ 0 & -0.906 & 0.179 & -0.876 & 0.331 \\ 1 & 0 & 0.906 & 0.324 & -0.137 \end{bmatrix} \begin{bmatrix} 39.566 & 0 & 1 \\ 50.522 & -0.906 & 0 \\ 68.164 & 0.179 & 0.906 \\ -1.282 & -0.876 & 0.324 \\ 0 & 0.331 & -0.137 \end{bmatrix} \tag{6.168}$$

$$JJ^T = \begin{bmatrix} 8765.915 & -32.449 & 100.907 \\ -32.449 & 1.730 & -0.167 \\ 100.907 & -0.167 & 1.945 \end{bmatrix} \tag{6.169}$$

The second step determines matrix r by solving the set of equations specified by Equation (6.145).

$$JJ^T r = d \tag{6.170}$$

$$\begin{bmatrix} 8765.915 & -32.449 & 100.907 \\ -32.449 & 1.730 & -0.167 \\ 100.907 & -0.167 & 1.945 \end{bmatrix} r = \begin{bmatrix} 64.286 \\ 0.107 \\ 1.085 \end{bmatrix} \tag{6.171}$$

$$\begin{bmatrix} 8765.915 & -32.449 & 100.907 \\ 0 & 1.610 & 0.207 \\ 0 & 0.207 & 0.783 \end{bmatrix} r = \begin{bmatrix} 64.286 \\ 0.345 \\ 0.345 \end{bmatrix} \tag{6.172}$$

$$\begin{bmatrix} 8765.915 & -32.449 & 100.907 \\ 0 & 1.610 & 0.207 \\ 0 & 0 & 0.756 \end{bmatrix} r = \begin{bmatrix} 64.286 \\ 0.345 \\ 0.301 \end{bmatrix} \tag{6.173}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} r = \begin{bmatrix} 3.356\text{E}-3 \\ 0.163 \\ 0.398 \end{bmatrix} \tag{6.174}$$

Lastly, the resultant r is substituted into Equation (6.158).

$$q' = J^T r \tag{6.175}$$

$$q' = \begin{bmatrix} 39.566 & 0 & 1 \\ 50.522 & -0.906 & 0 \\ 68.164 & 0.179 & 0.906 \\ -1.282 & -0.876 & 0.324 \\ 0 & 0.331 & -0.137 \end{bmatrix} \begin{bmatrix} 3.356\text{E}-3 \\ 0.163 \\ 0.398 \end{bmatrix} \tag{6.176}$$

$$q' = \begin{bmatrix} 0.531 \\ 0.022 \\ 0.619 \\ -0.018 \\ -5.730E-4 \end{bmatrix} \tag{6.177}$$

This result is verified by substituting it and the Jacobian of Equation (6.113) into Equation (6.127).

$$d = Jq \tag{6.178}$$

$$d = \begin{bmatrix} 260.692 & 23.559 & 166.988 & 35.123 & 0 \\ 39.566 & 50.522 & 68.164 & -1.282 & 0 \\ 0 & 219.546 & -61.716 & 93.620 & 0 \\ 0 & -0.906 & 0.179 & -0.876 & 0.331 \\ 0 & 0.423 & 0.383 & -0.358 & -0.934 \\ 1 & 0 & 0.906 & 0.324 & -0.137 \end{bmatrix} \begin{bmatrix} 0.531 \\ 0.022 \\ 0.619 \\ -0.018 \\ -5.730E-4 \end{bmatrix} \tag{6.179}$$

$$d = \begin{bmatrix} 241.679 \\ 64.338 \\ -35.057 \\ 0.106 \\ 0.253 \\ 1.086 \end{bmatrix} \tag{6.180}$$

Comparison with the desired set of rates in Equation (6.166) shows that there is no appreciable difference between any of the three pair. The three remaining rates, stated in Equation (6.112), differ widely. The Euclidean norm of this solution is less than that of the original, as found in Equation (6.163) to be 1.083.

$$|q'| = <0.531\ \ 0.022\ \ 0.619\ \ -0.018\ \ -5.730E-4> \cdot <0.531\ \ 0.022\ \ 0.619\ \ -0.018\ \ -5.730E-4> \quad (6.181)$$

$$|q'| = 0.665 \quad (6.182)$$

As in the first example of the under-determined case, the desired rates are met at overall lower joint rates than the original solution.

c. Program Control. The procedure `under_determined_case` differs from that of the over-determined case only in the order of execution; the steps may be seen to be the same.

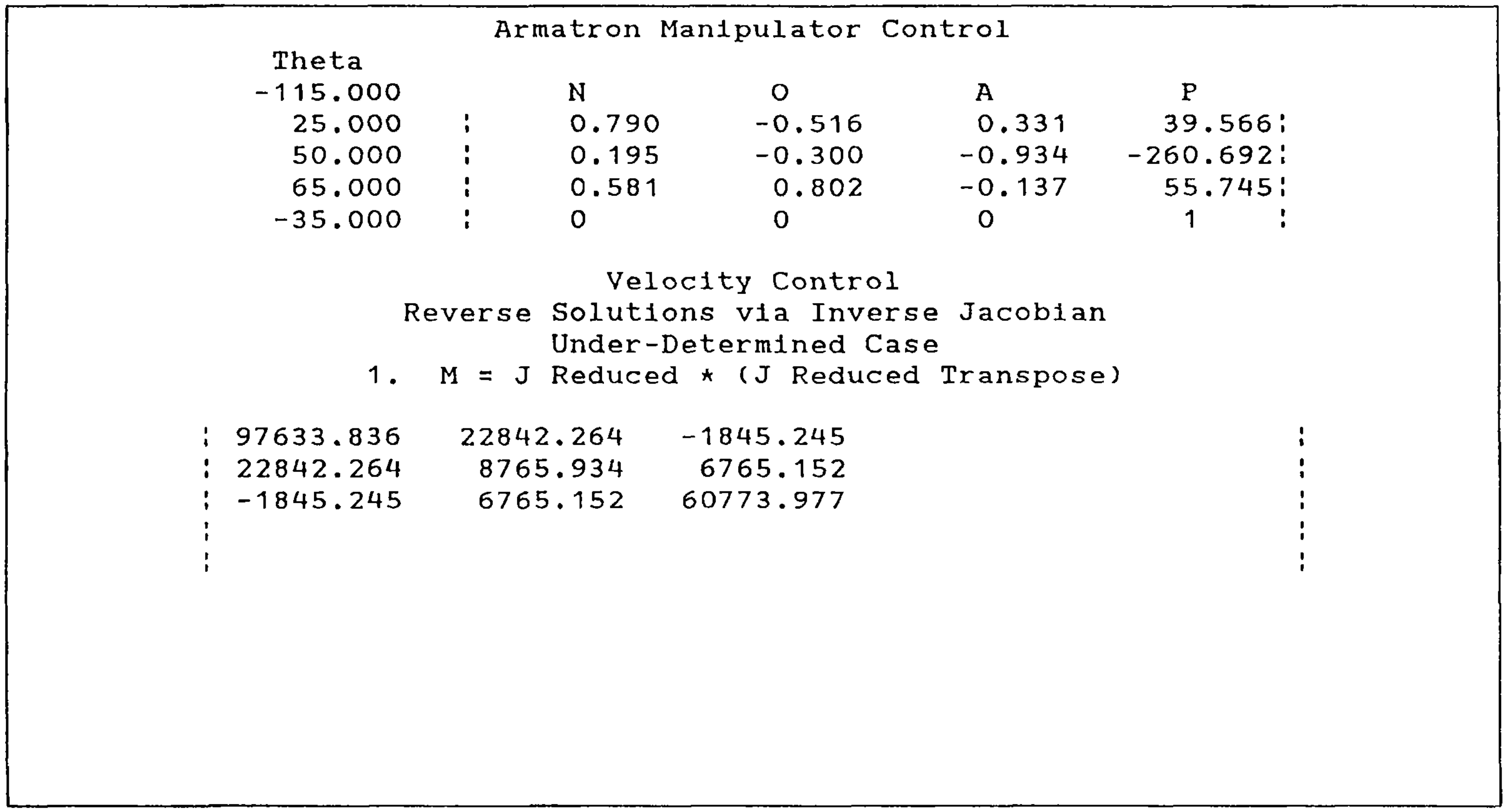
```
subhead = "1.  M = J Reduced * (J Reduced Transpose)";
matrix_by_matrix (total, jacobian_reduced, m, subhead);
for (i = 1; i <= total; i++)
   dtr_5[i] = delta_tr_reduced[i];
subhead = "2.  Solve M * Y = d trans/rot rates";
inconsistent = solve_simul_eqns_myv (m, dtr_5, total, y, subhead);
if ( inconsistent == 0 )
   {
   subhead = "3.  d Theta = (J Reduced Transpose) * Y";
   matrix_by_vector (jacobian_reduced, total, y, delta_theta,
                     subhead);
   }
return (inconsistent);
```

As the simultaneous equations are solved as the second step instead of the third, the third step here, multiplication of the involved matrix and vector, is omitted altogether if the second step fails due to inconsistency in the equations.

d. Program Examples. Figures 6.8 through 6.12 detail the screens displayed during the solution process of the under-determined case for the selection of the translation rates as input. The products obtained in Figures 6.9 and 6.10 closely parallel the results obtained in Equations 6.150 and 6.155. The final rates obtained in Figure 6.11 are almost precisely those of Equation (6.158).

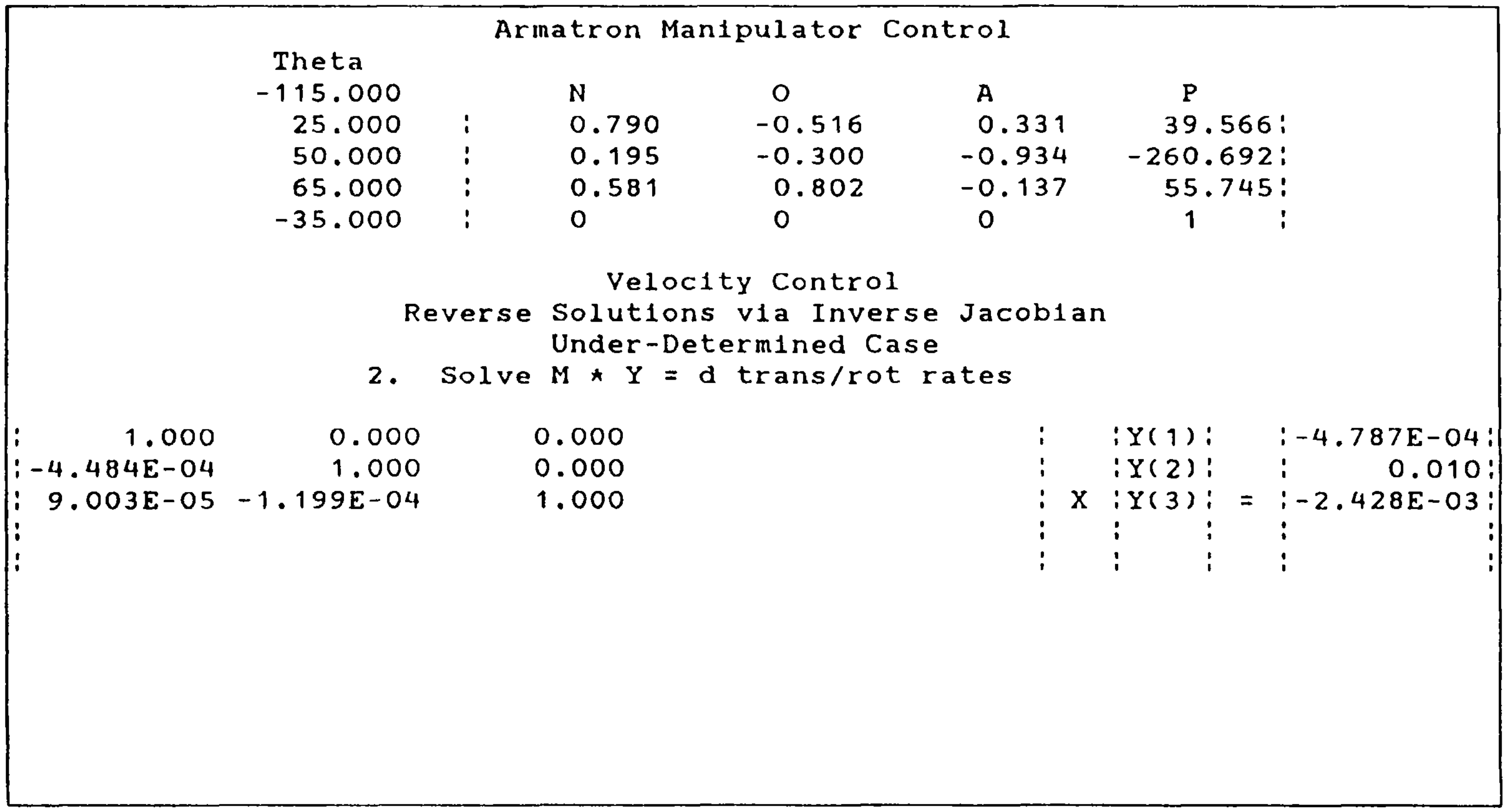
```
                         Armatron Manipulator Control
             Theta
            -115.000            N          O          A          P
              25.000    :     0.790     -0.516      0.331      39.566:
              50.000    :     0.195     -0.300     -0.934    -260.692:
              65.000    :     0.581      0.802     -0.137      55.745:
             -35.000    :     0                0          0            1       :

                              Velocity Control
                     Reverse Solutions via Inverse Jacobian

Delta Rates                        Jacobian
:    196.548:     :   260.692     23.559    166.988     35.123      0.000: :d T1:
:     64.286:     :    39.566     50.522     68.164     -1.282      0.000: :d T2:
:    -75.924:  =  :     0.000    219.546    -61.716     93.620      0.000:X:d T3:
:    unused :     :     0.000     -0.906      0.179     -0.876      0.331: :d T4:
:    unused :     :     0.000      0.423      0.383     -0.358     -0.934: :d T5:
:    unused :     :     1.000      0.000      0.906      0.324     -0.137:
```

Figure 6.8. Inverse Jacobian, Under-Determined Example 1: Inputs



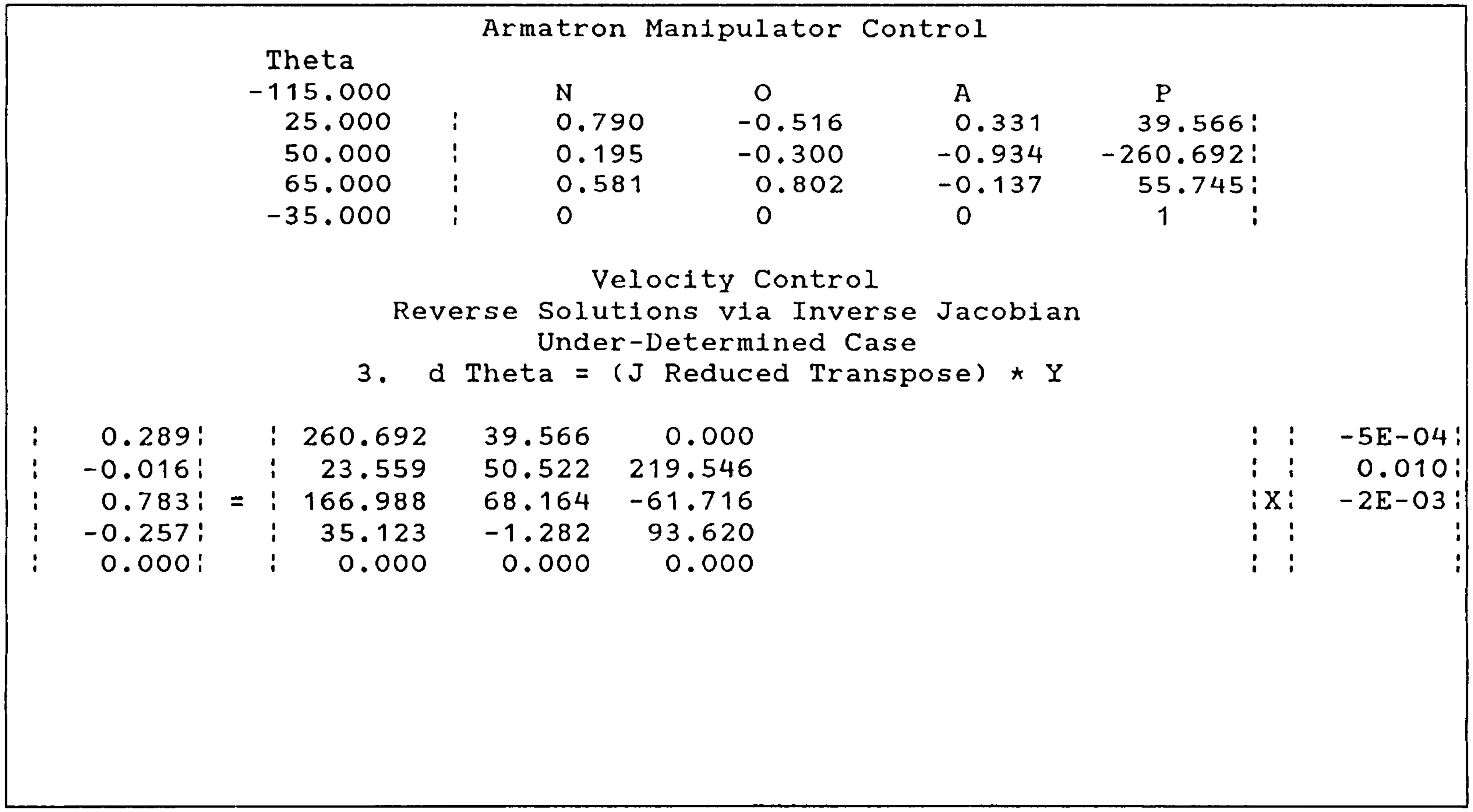
```
                    Armatron Manipulator Control
      Theta
     -115.000            N          O          A          P
       25.000   :     0.790     -0.516      0.331     39.566:
       50.000   :     0.195     -0.300     -0.934   -260.692:
       65.000   :     0.581      0.802     -0.137     55.745:
      -35.000   :     0       0          0          1      :

                         Velocity Control
                Reverse Solutions via Inverse Jacobian
                        Under-Determined Case
              1.   M = J Reduced * (J Reduced Transpose)

  : 97633.836   22842.264   -1845.245                          :
  : 22842.264    8765.934    6765.152                          :
  : -1845.245    6765.152   60773.977                          :
  :                                                            :
  :                                                            :
```

Figure 6.9. Inverse Jacobian, Under-Determined Example 1: Step 1



```
                           Armatron Manipulator Control
                  Theta
                 -115.000            N           O           A           P
                   25.000    :     0.790      -0.516       0.331      39.566:
                   50.000    :     0.195      -0.300      -0.934    -260.692:
                   65.000    :     0.581       0.802      -0.137      55.745:
                  -35.000    :     0                0           0           1     :

                                 Velocity Control
                       Reverse Solutions via Inverse Jacobian
                              Under-Determined Case
                         2.  Solve M * Y = d trans/rot rates

:      1.000       0.000       0.000                          :    :Y(1):    :-4.787E-04:
:-4.484E-04       1.000       0.000                          :    :Y(2):    :     0.010:
: 9.003E-05 -1.199E-04       1.000                          :  X :Y(3):  = :-2.428E-03:
:                                                            :    :    :    :          :
:                                                            :    :    :    :          :
```

Figure 6.10. Inverse Jacobian, Under-Determined Example 1: Step 2



```
                         Armatron Manipulator Control
              Theta
            -115.000             N           O           A           P
              25.000    :      0.790      -0.516       0.331      39.566:
              50.000    :      0.195      -0.300      -0.934    -260.692:
              65.000    :      0.581       0.802      -0.137      55.745:
             -35.000    :      0                0           0           1     :

                               Velocity Control
                      Reverse Solutions via Inverse Jacobian
                              Under-Determined Case
                    3.   d Theta = (J Reduced Transpose) * Y

:     0.289:     : 260.692   39.566    0.000                            : :    -5E-04:
:    -0.016:     :  23.559   50.522  219.546                            : :     0.010:
:     0.783:  =  : 166.988   68.164  -61.716                            :X:    -2E-03:
:    -0.257:     :  35.123   -1.282   93.620                            : :          :
:     0.000:     :   0.000    0.000    0.000                            : :          :
```

Figure 6.11. Inverse Jacobian, Under-Determined Example 1: Step 3

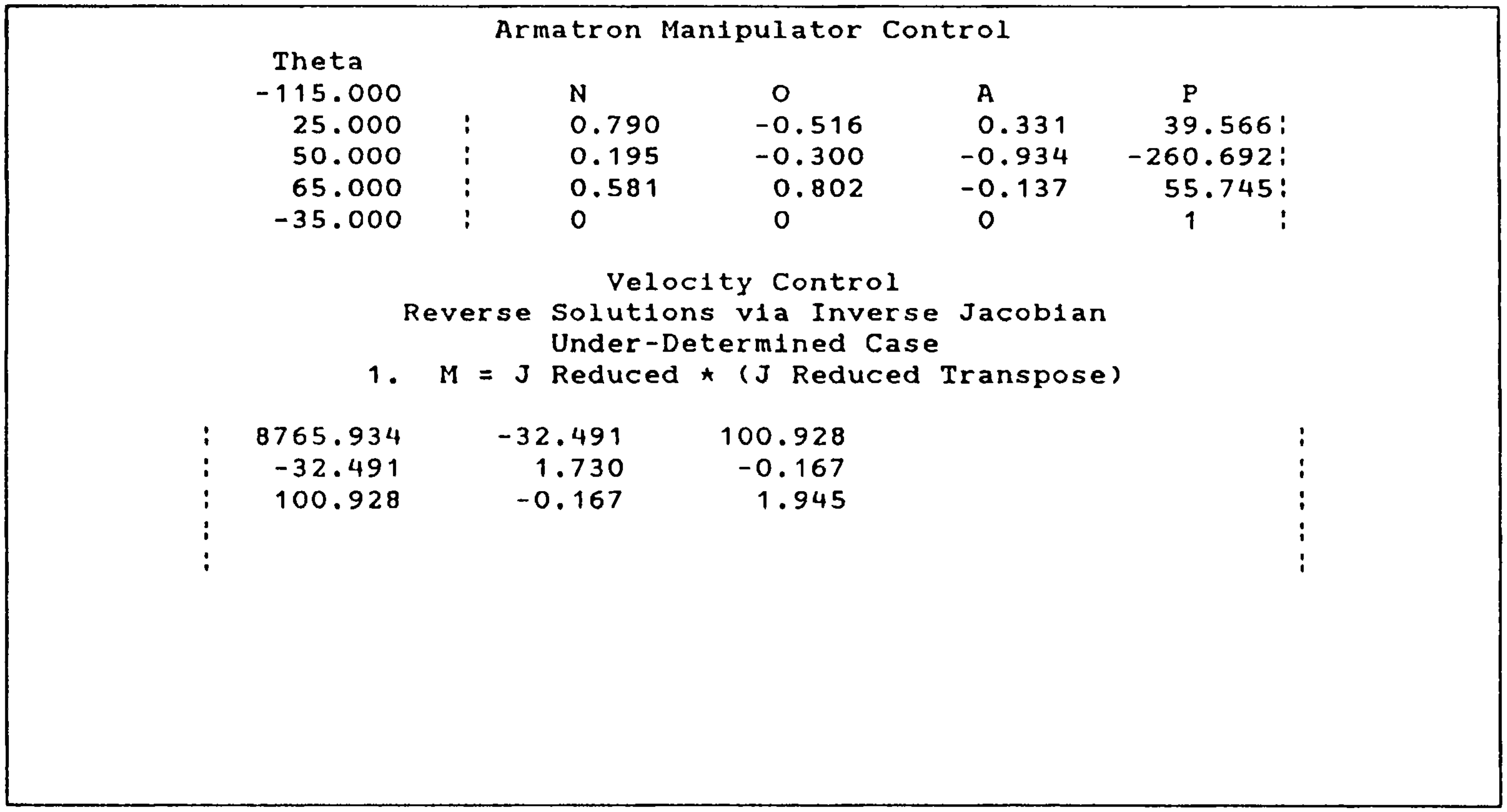
```
                  Armatron Manipulator Control
        Theta
       -115.000          N          O          A          P
         25.000   ¦    0.790     -0.516      0.331     39.566¦
         50.000   ¦    0.195     -0.300     -0.934   -260.692¦
         65.000   ¦    0.581      0.802     -0.137     55.745¦
        -35.000   ¦    0          0          0          1     ¦

                        Velocity Control
                Reverse Solutions via Inverse Jacobian
                     Under-Determined Case
             Input: Delta Translational & Rotational Rates
                     ¦dtx¦      ¦   196.548¦
                     ¦dty¦      ¦    64.286¦
                     ¦dtz¦  =   ¦   -75.924¦
                     ¦drx¦      ¦   unused ¦
                     ¦dry¦      ¦   unused ¦
                     ¦drz¦      ¦   unused ¦
                   Output:  Delta Theta Rates
                     ¦dT1¦      ¦     0.289¦
                     ¦dT2¦      ¦    -0.016¦
                     ¦dT3¦  =   ¦     0.783¦
                     ¦dT4¦      ¦    -0.257¦
                     ¦dT5¦      ¦     0.000¦
```

Figure 6.12. Inverse Jacobian, Under-Determined Example 1: Results

The solution of the under-determined case by the program specified in the second numerical example is depicted in Figures 6.13 through 6.17. The results show in Figures 6.14, 6.15, and 6.16 may be compared with those of Equations (6.169), (6.174), and (6.177), respectively, to see that there are no significant differences.

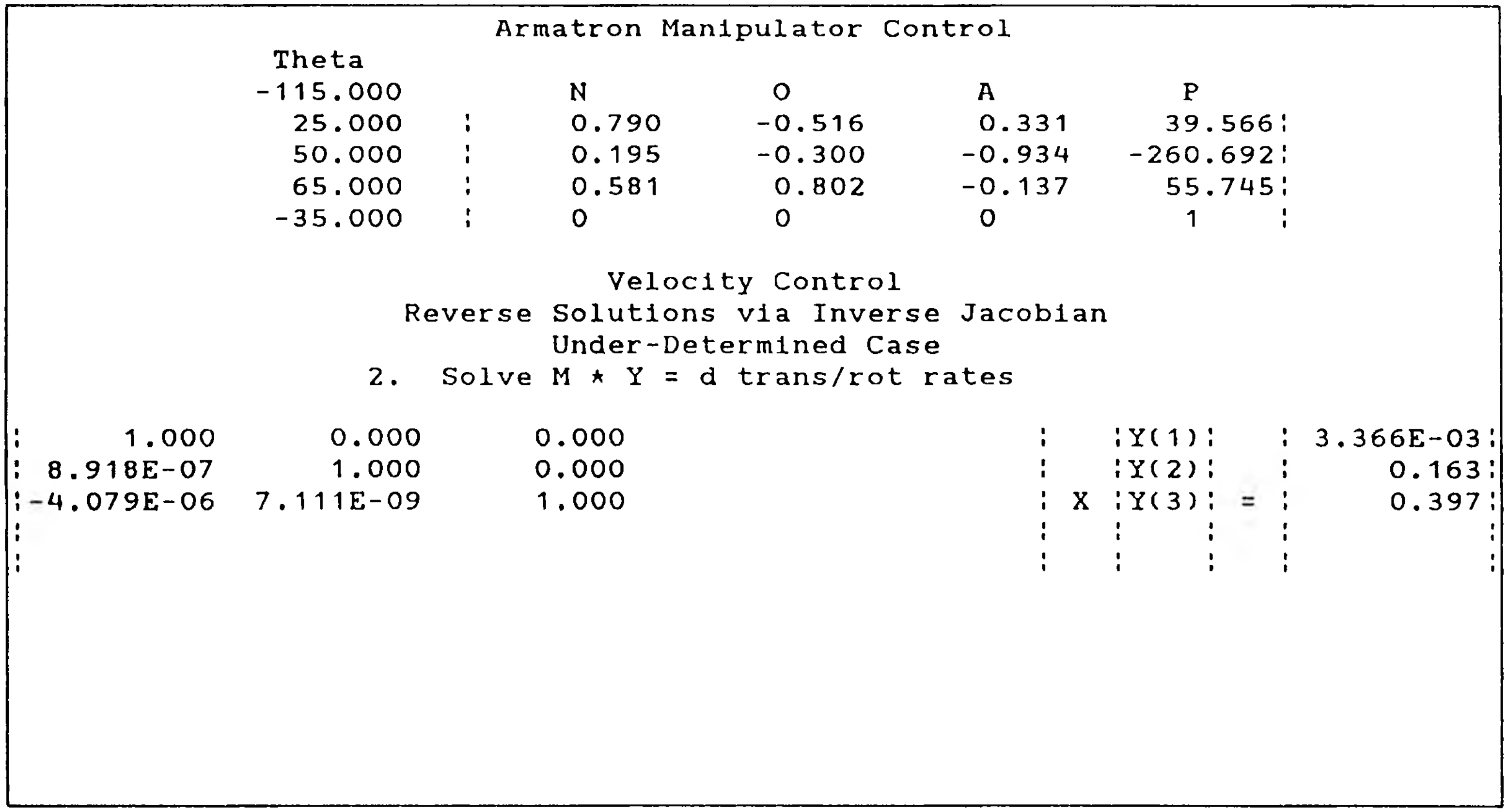
```
                    Armatron Manipulator Control
       Theta
     -115.000            N          O          A          P
       25.000   :     0.790     -0.516      0.331     39.566:
       50.000   :     0.195     -0.300     -0.934   -260.692:
       65.000   :     0.581      0.802     -0.137     55.745:
      -35.000   :     0              0          0      1     :

                          Velocity Control
                 Reverse Solutions via Inverse Jacobian

Delta Rates                    Jacobian
:   unused :   :  260.692    23.559    166.988    35.123     0.000: :d T1:
:   64.286:    :   39.566    50.522     68.164    -1.282     0.000: :d T2:
:   unused : = :    0.000   219.546    -61.716    93.620     0.000:X:d T3:
:    0.107:    :    0.000    -0.906      0.179    -0.876     0.331: :d T4:
:   unused :   :    0.000     0.423      0.383    -0.358    -0.934: :d T5:
:    1.085:    :    1.000     0.000      0.906     0.324    -0.137:
```

Figure 6.13. Inverse Jacobian, Under-Determined Example 2: Inputs



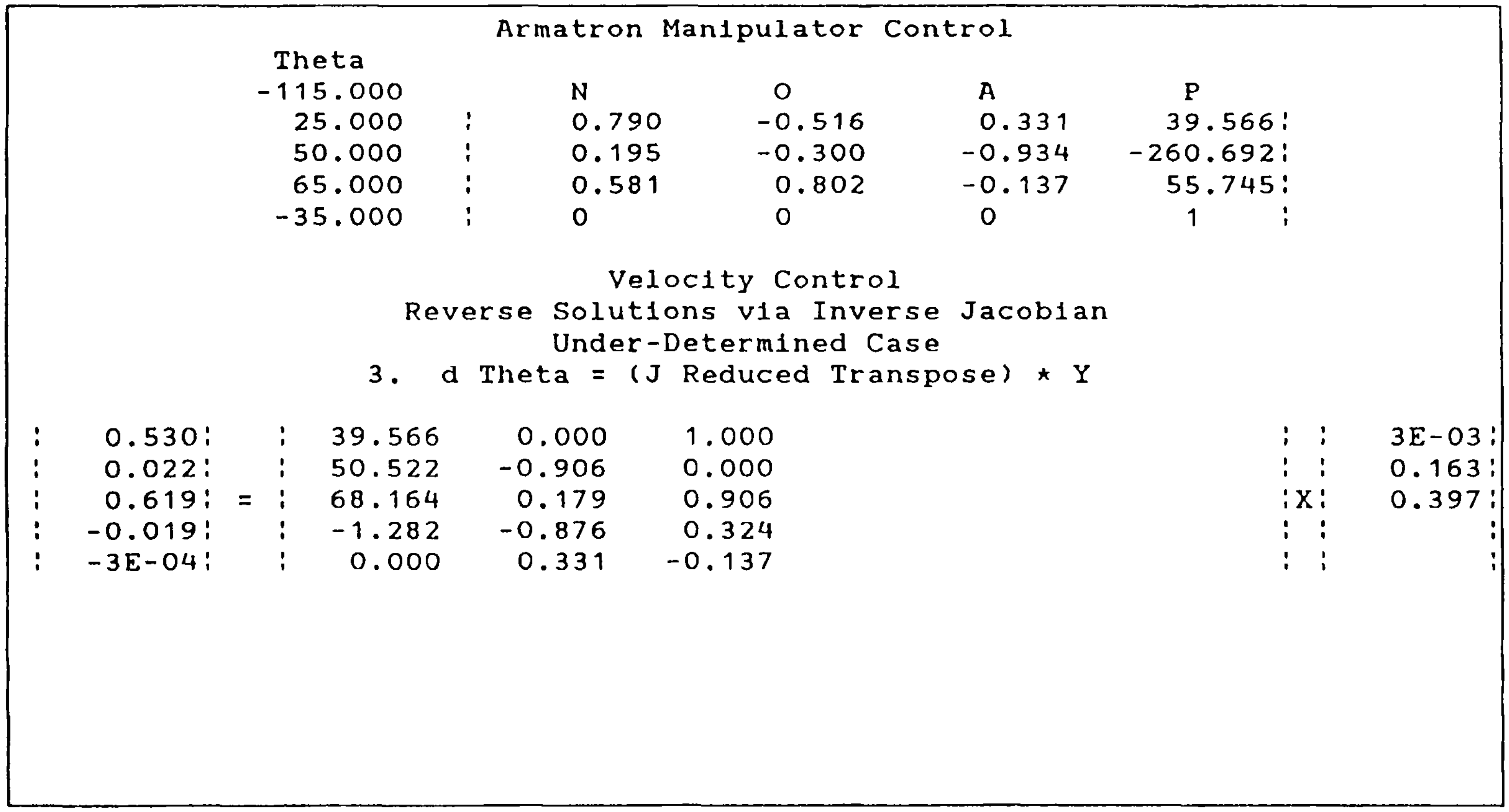
```
                      Armatron Manipulator Control
       Theta
      -115.000            N          O          A          P
        25.000   :      0.790     -0.516      0.331      39.566:
        50.000   :      0.195     -0.300     -0.934    -260.692:
        65.000   :      0.581      0.802     -0.137      55.745:
       -35.000   :      0               0          0          1    :

                            Velocity Control
                  Reverse Solutions via Inverse Jacobian
                          Under-Determined Case
                  1.   M = J Reduced * (J Reduced Transpose)

  :   8765.934      -32.491      100.928                                :
  :    -32.491        1.730       -0.167                                :
  :    100.928       -0.167        1.945                                :
  :                                                                     :
  :                                                                     :
```

Figure 6.14. Inverse Jacobian, Under-Determined Example 2: Step 1



```
                         Armatron Manipulator Control
               Theta
              -115.000            N          O          A          P
                25.000   :     0.790     -0.516      0.331     39.566:
                50.000   :     0.195     -0.300     -0.934   -260.692:
                65.000   :     0.581      0.802     -0.137     55.745:
               -35.000   :     0             0          0          1      :

                                  Velocity Control
                      Reverse Solutions via Inverse Jacobian
                              Under-Determined Case
                      2.   Solve M * Y = d trans/rot rates

:      1.000        0.000        0.000                            :     :Y(1):    : 3.366E-03:
: 8.918E-07        1.000        0.000                            :     :Y(2):    :      0.163:
:-4.079E-06  7.111E-09        1.000                            :  X  :Y(3):  =  :      0.397:
:                                                                 :     :    :    :          :
:                                                                 :     :    :    :          :
```

Figure 6.15. Inverse Jacobian, Under-Determined Example 2: Step 2



```
                    Armatron Manipulator Control
         Theta
        -115.000           N          O          A          P
          25.000   :     0.790     -0.516      0.331     39.566:
          50.000   :     0.195     -0.300     -0.934   -260.692:
          65.000   :     0.581      0.802     -0.137     55.745:
         -35.000   :     0           0          0         1    :

                          Velocity Control
                Reverse Solutions via Inverse Jacobian
                       Under-Determined Case
             3.   d Theta = (J Reduced Transpose) * Y

:    0.530:   :  39.566    0.000     1.000                    : :   3E-03:
:    0.022:   :  50.522   -0.906     0.000                    : :   0.163:
:    0.619: = :  68.164    0.179     0.906                    :X:   0.397:
:   -0.019:   :  -1.282   -0.876     0.324                    : :        :
:   -3E-04:   :   0.000    0.331    -0.137                    : :        :
```

Figure 6.16. Inverse Jacobian, Under-Determined Example 2: Step 3

```
                     Armatron Manipulator Control
      Theta
     -115.000              N            O            A            P
       25.000    :       0.790       -0.516        0.331       39.566:
       50.000    :       0.195       -0.300       -0.934     -260.692:
       65.000    :       0.581        0.802       -0.137       55.745:
      -35.000    :       0                  0            0            1     :

                              Velocity Control
                  Reverse Solutions via Inverse Jacobian
                          Under-Determined Case
                Input: Delta Translational & Rotational Rates
                           :dtx:       :   unused :
                           :dty:       :    64.286:
                           :dtz:   =   :   unused :
                           :drx:       :     0.107:
                           :dry:       :   unused :
                           :drz:       :     1.085:
                         Output:  Delta Theta Rates
                           :dT1:       :     0.530:
                           :dT2:       :     0.022:
                           :dT3:   =   :     0.619:
                           :dT4:       :    -0.019:
                           :dT5:       :-3.286E-04:
```

Figure 6.17. Inverse Jacobian, Under-Determined Example 2: Results

C. JOINT RATES BY DIFFERENTIATION

The previous section determined the joint variable rates necessary to obtain a particular set of translational and rotational rates of the manipulator end coordinate frame by solving the Jacobian equation for the end frame rates by numerical means. An alternative to this method is to differentiate the joint variable solutions obtained in Chapter 5; this will result in relationships expressing the joint rates directly.

1. Derivation of Equations.

Recall from Chapter 5 that the joint variable values needed to obtain a particular orientation and position of the manipulator end coordinate frame were derived in terms of that orientation and position. Each of the specific equations found there may be differentiated to obtain a joint variable rate in terms of the rates of change of the orientation vectors $\vec{n}$, $\vec{o}$, and $\vec{a}$ and position vector $\vec{p}$. The rate of change of the position vector is known as its components are given by the desired translational rates, ${}^0d_x$, ${}^0d_y$, and ${}^0d_z$. The rates of change of the orientation vectors must be determined from known information, which in this case is the orientation vector directions for some specified set of joint variables and a set of desired translational and rotational rates.

a. Rates of Change of the Orientation Vectors.

Recall that the orientation and position matrix T is the result of the sequence of A matrix transformations taking the triple of unit vectors at the base coordinate frame to those of the manipulator end frame. Each A_i matrix is a transformation with respect to the $i-1$ coordinate frame. Consider a sequence of rotations $Rot(\delta_x, \delta_y, \delta_z)$ where each delta is a small rotation about its respective axis. Each successive rotation shall be made with respect to the coordinate frame resulting from the previous rotation. Consider first carrying out the rotations in x-y-z order as stated.

$$Rot(\delta_x, \delta_y, \delta_z) = Rot(x, \delta_x)Rot(y, \delta_y)Rot(z, \delta_z) \tag{6.183}$$

$$Rot(\delta_x, \delta_y, \delta_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\delta_x & -\sin\delta_x & 0 \\ 0 & \sin\delta_x & \cos\delta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\delta_y & 0 & \sin\delta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\delta_y & 0 & \cos\delta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\delta_z & -\sin\delta_z & 0 & 0 \\ \sin\delta_z & \cos\delta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.184}$$

The rotations δ_x, δ_y, and δ_z are very small angles. From the calculus,

$$\lim_{t\to 0} \frac{\sin t}{t} = 1 \tag{6.185}$$

$$\lim_{t\to 0} \frac{1 - \cos t}{t} = 0 \tag{6.186}$$

Approximations may then be made for the δ angles.

$$\sin\delta = \delta \tag{6.187}$$

$$\cos\delta = 1 \tag{6.188}$$

These substitutions are then made in Equation (6.184).

$$Rot(\delta_x, \delta_y, \delta_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -\delta_x & 0 \\ 0 & \delta_x & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \delta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\delta_y & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\delta_z & 0 & 0 \\ \delta_z & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.189}$$

$$Rot(\delta_x, \delta_y, \delta_z) = \begin{bmatrix} 1 & 0 & \delta_y & 0 \\ \delta_x\delta_y & 1 & -\delta_x & 0 \\ -\delta_y & \delta_x & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\delta_z & 0 & 0 \\ \delta_z & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.190}$$

$$Rot(\delta_x, \delta_y, \delta_z) = \begin{bmatrix} 1 & -\delta_z & \delta_y & 0 \\ \delta_x\delta_y + \delta_z & -\delta_x\delta_y + 1 & -\delta_x & 0 \\ -\delta_y + \delta_x\delta_z & \delta_y\delta_z + \delta_x & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.191}$$

Since each of the rotations is very small, the product of any two of them is small enough to be considered insignificant and may thus be replaced by zero.

$$Rot(\delta_x, \delta_y, \delta_z) = \begin{bmatrix} 1 & -\delta_z & \delta_y & 0 \\ \delta_z & 1 & -\delta_x & 0 \\ -\delta_y & \delta_x & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.192}$$

Exhaustive arrangements of the matrices will show that any ordering of the three rotations yields the same final result.

Consider now the effect the rotations ${}^0\delta_x$, ${}^0\delta_y$, and ${}^0\delta_z$ have upon the orientation of the coordinate frame of the manipulator end. This change shall be defined *dT*. As matrix $Rot({}^0\delta_x, {}^0\delta_y, {}^0\delta_z)$ is defined with respect to the base coordinate frame, it must be pre-multiplied by matrix *T*.

$$T + dT = Rot({}^0\delta_x, {}^0\delta_y, {}^0\delta_z)T \tag{6.193}$$

$$dT = Rot({}^0\delta_x, {}^0\delta_y, {}^0\delta_z)T - T \tag{6.194}$$

$$dT = \left[\begin{bmatrix} 1 & -{}^0\delta_z & {}^0\delta_y & 0 \\ {}^0\delta_z & 1 & -{}^0\delta_x & 0 \\ -{}^0\delta_y & {}^0\delta_x & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \right] T \tag{6.195}$$

$$d \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -{}^0\delta_z & {}^0\delta_y & 0 \\ {}^0\delta_z & 0 & -{}^0\delta_x & 0 \\ -{}^0\delta_y & {}^0\delta_x & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.196}$$

By reducing the amount of time over which the delta quantities in equations such as this are defined, instantaneous velocities and hence derivatives are obtained. Vectors ${}^0\bar{\delta}$ and ${}^0\bar{d}$ are understood to be instantaneous velocities from this point on. Thus, equating elements in column 1,

$$\frac{d}{dt} n_x = -{}^0\delta_z n_y + {}^0\delta_y n_z \tag{6.197}$$

$$\frac{d}{dt} n_y = {}^0\delta_z n_x - {}^0\delta_x n_z \tag{6.198}$$

$$\frac{d}{dt} n_z = -{}^0\delta_y n_x + {}^0\delta_x n_y \tag{6.199}$$

The equations for the elements of the second and third dT columns parallel these; for example,

$$\frac{d}{dt} o_x = -{}^0\delta_z o_y + {}^0\delta_y o_z \tag{6.200}$$

The effect of rotational rates ${}^0\delta_x$, ${}^0\delta_y$, and ${}^0\delta_z$ on the position specified by the fourth column are ignored here as the translational rates are explicitly given by ${}^0d_x$, ${}^0d_y$, and ${}^0d_z$.

b. Joint Variable 3. The derivation of equations in Chapter 5 led first to a relationship for the cosine of joint 3's variable, θ_3, in Equation (5.81).

$$c_3 = \frac{p_{a_x}^2 + p_{a_y}^2 + p_{a_z}^2 - a_3^2 - a_2^2}{2a_2a_3} \tag{6.201}$$

Differentiation yields a formula for the rate of θ_3 in terms of the rate at which the end of the arm proper is moving.

$$-s_3\frac{d}{dt}\theta_3 = \frac{2p_{a_x}\frac{d}{dt}p_{a_x} + 2p_{a_y}\frac{d}{dt}p_{a_y} + 2p_{a_z}\frac{d}{dt}p_{a_z}}{2a_2a_3} \tag{6.202}$$

$$\frac{d}{dt}\theta_3 = \frac{p_{a_x}\frac{d}{dt}p_{a_x} + p_{a_y}\frac{d}{dt}p_{a_y} + p_{a_z}\frac{d}{dt}p_{a_z}}{-a_2a_3s_3} \tag{6.203}$$

Equations (5.41), (5.42), and (5.43) defined the arm vector components.

$$p_{a_x} = p_x - p_{w_x} \tag{6.204}$$

$$p_{a_y} = p_y - p_{w_y} \tag{6.205}$$

$$p_{a_z} = p_z - p_{w_z} \tag{6.206}$$

Equations (5.38), (5.39), and (5.40) had previously defined the wrist vector components.

$$p_{w_x} = d_5a_x \tag{6.207}$$

$$p_{w_y} = d_5 a_y \tag{6.208}$$

$$p_{w_z} = d_5 a_z \tag{6.209}$$

These three expressions are substituted into their respective positions in Equations (6.204), (6.205), and (6.206) prior to differentiation.

$$p_{a_x} = p_x - d_5 a_x \tag{6.210}$$

$$\frac{d}{dt} p_{a_x} = \frac{d}{dt} p_x - d_5 \frac{d}{dt} a_x \tag{6.211}$$

$$p_{a_y} = p_y - d_5 a_y \tag{6.212}$$

$$\frac{d}{dt} p_{a_y} = \frac{d}{dt} p_y - d_5 \frac{d}{dt} a_y \tag{6.213}$$

$$p_{a_z} = p_z - d_5 a_z \tag{6.214}$$

$$\frac{d}{dt} p_{a_z} = \frac{d}{dt} p_z - d_5 \frac{d}{dt} a_z \tag{6.215}$$

The differential relationships in Equations (6.211), (6.213), and (6.215) will be employed in the solutions for the rates of θ_1 and θ_2 also; thus, in order that they only be calculated once, they shall not be substituted into any other equations.

Returning now to Equation (6.203), it can be seen that the rate of the joint variable is defined in terms of the rate of change of the orientation and position matrix T, except when the sine of θ_3 becomes zero. This situation occurs when the joint variable is itself zero and the elbow is fully extended; unfortunately, no other solution is immediately available, so this instance is treated as a singularity.

c. Joint Variable 1. Subsequent to the derivation of an equation involving θ_3, the following relationship was established for θ_1 in Equation (5.90).

$$-p_{a_x}s_1 + p_{a_y}c_1 = a_3s_3 \tag{6.216}$$

Clearly, the rate at which θ_1 turns shall be dependent in part on that of θ_3. Differentiating,

$$-p_{a_x}c_1 \frac{d}{dt}\theta_1 - s_1 \frac{d}{dt}p_{a_x} - p_{a_y}s_1 \frac{d}{dt}\theta_1 + c_1 \frac{d}{dt}p_{a_y} = a_3c_3 \frac{d}{dt}\theta_3 \tag{6.217}$$

$$-(p_{a_x}c_1 + p_{a_y}s_1)\frac{d}{dt}\theta_1 = a_3c_3 \frac{d}{dt}\theta_3 + s_1 \frac{d}{dt}p_{a_x} - c_1 \frac{d}{dt}p_{a_y} \tag{6.218}$$

$$\frac{d}{dt}\theta_1 = \frac{c_1 \frac{d}{dt}p_{a_y} - s_1 \frac{d}{dt}p_{a_x} - a_3c_3 \frac{d}{dt}\theta_3}{p_{a_x}c_1 + p_{a_y}s_1} \tag{6.219}$$

As was the case for θ_3, only one equation was obtained during the derivation of a solution for θ_1. Thus, should the denominator of Equation (6.219) become zero, no other solution exists and the situation must thus be treated as a singularity.

d. Joint Variable 2. The remaining arm variable, θ_2, differs from the previous two in that two equations were obtained dealing with it in Chapter 5. The first was Equation (5.112) and has its derivative taken here to provide one formula for the joint rate.

$$p_{a_x}c_1 + p_{a_y}s_1 = c_2(a_3c_3 + a_2) \tag{6.220}$$

$$\begin{aligned}\left(\frac{d}{dt}p_{a_x}\right)c_1 + p_{a_x}\left(-s_1\frac{d}{dt}\theta_1\right) + \left(\frac{d}{dt}p_{a_y}\right)s_1 + p_{a_y}\left(c_1\frac{d}{dt}\theta_1\right) = \\ -s_2\left(\frac{d}{dt}\theta_2\right)(a_3c_3 + a_2) + c_2\left(-a_3s_3\frac{d}{dt}\theta_3\right)\end{aligned} \tag{6.221}$$

$$-s_2(a_3c_3 + a_2)\frac{d}{dt}\theta_2 = \\ c_1p_{a_y}\frac{d}{dt}\theta_1 - s_1p_{a_x}\frac{d}{dt}\theta_1 + c_1\frac{d}{dt}p_{a_x} + s_1\frac{d}{dt}p_{a_y} + a_3c_2s_3\frac{d}{dt}\theta_3 \tag{6.222}$$

$$\frac{d}{dt}\theta_2 = \frac{(c_1p_{a_y} - s_1p_{a_x})\frac{d}{dt}\theta_1 + c_1\frac{d}{dt}p_{a_x} + s_1\frac{d}{dt}p_{a_y} + a_3c_2s_3\frac{d}{dt}\theta_3}{-s_2(a_3c_3 + a_2)} \tag{6.223}$$

The division in this formula is undefined when either s_2 or $a_3c_3 + a_2$ becomes zero. As explained in Chapter 5, $a_3c_3 + a_2$ cannot become zero as this requires θ_3 to take on the value of -180°, for which the arm folds back on itself. This situation is physically impossible and would be detected during examination of the joint range. Factor s_2 can become zero and does when θ_2 is 0°, a value which yields a level arm. This situation need not be treated as a singularity, however, because of the existence of a second equation involving θ_2, Equation (5.114); it has its derivative taken here.

$$p_{a_z} = s_2(a_3c_3 + a_2) \tag{6.224}$$

$$\frac{d}{dt}p_{a_z} = \left(c_2\frac{d}{dt}\theta_2\right)(a_3c_3 + a_2) + s_2\left(-a_3s_3\frac{d}{dt}\theta_3\right) \tag{6.225}$$

$$\frac{d}{dt}\theta_2 = \frac{\frac{d}{dt}p_{a_z} + a_3s_2s_3\frac{d}{dt}\theta_3}{c_2(a_3c_3 + a_2)} \tag{6.226}$$

The division here is undefined whenever c_2 or $a_3c_3 + a_2$ becomes zero. As with the first formula, $a_3c_3 + a_2$ will be prevented from becoming zero. The other factor, c_2, becomes zero when θ_2 is 90°, where the first formula became zero for a θ_2 of 0°. Physically, θ_2 is limited to a range of -5° to 30°, so this second formula should be viable in all practical situations. Both formulas shall be utilized to allow for as many situations to be considered as possible without regard to the manipulator's physical limitations.

e. Joint Variable 4. The equations derived in Chapter 5 dealing with wrist variables θ_4 and θ_5 are independent of one another and thus may be examined in either order. There are two equations available for each joint, so no singularities need be dealt with here. Equation (5.183) was the first arrived at for θ_4 and is differentiated here.

$$c_4 = a_x c_1 s_2 + a_y s_1 s_2 - a_z c_2 \tag{6.227}$$

$$\begin{aligned} -s_4 \frac{d}{dt}\theta_4 = \left(\frac{d}{dt}a_x\right)c_1 s_2 + a_x \frac{d}{dt}(c_1 s_2) + \left(\frac{d}{dt}a_y\right)s_1 s_2 \\ + a_y \frac{d}{dt}(s_1 s_2) - \left(\frac{d}{dt}a_z\right)c_2 - a_z\left(-s_2 \frac{d}{dt}\theta_2\right) \end{aligned} \tag{6.228}$$

$$\begin{aligned} -s_4 \frac{d}{dt}\theta_4 = c_1 s_2 \frac{d}{dt}a_x + a_x\left(-s_1\left(\frac{d}{dt}\theta_1\right)s_2 + c_1 c_2\left(\frac{d}{dt}\theta_2\right)\right) + s_1 s_2 \frac{d}{dt}a_y \\ + a_y\left(c_1\left(\frac{d}{dt}\theta_1\right)s_2 + s_1 c_2\left(\frac{d}{dt}\theta_2\right)\right) - c_2 \frac{d}{dt}a_z + a_z s_2 \frac{d}{dt}\theta_2 \end{aligned} \tag{6.229}$$

$$\begin{aligned} -s_4 \frac{d}{dt}\theta_4 = c_1 s_2 \frac{d}{dt}a_x - a_x s_1 s_2 \frac{d}{dt}\theta_1 + a_x c_1 c_2 \frac{d}{dt}\theta_2 + s_1 s_2 \frac{d}{dt}a_y \\ + a_y c_1 s_2 \frac{d}{dt}\theta_1 + a_y s_1 c_2 \frac{d}{dt}\theta_2 - c_2 \frac{d}{dt}a_z + a_z s_2 \frac{d}{dt}\theta_2 \end{aligned} \tag{6.230}$$

$$\begin{aligned} \frac{d}{dt}\theta_4 = \Bigg((a_x s_1 - a_y c_1)s_2 \frac{d}{dt}\theta_1 - ((a_x c_1 + a_y s_1)c_2 + a_z s_2)\frac{d}{dt}\theta_2 \\ - c_1 s_2 \frac{d}{dt}a_x - s_1 s_2 \frac{d}{dt}a_y + c_2 \frac{d}{dt}a_z\Bigg)/s_4 \end{aligned} \tag{6.231}$$

In the event that θ_4 is 0° or 180°, causing its sine in the denominator of the formula above to become zero, the formula for the rate of θ_4 obtained from Equation (5.184) may be employed.

$$s_4 = a_x(c_1 c_2 c_3 - s_1 s_3) + a_y(s_1 c_2 c_3 + c_1 s_3) + a_z s_2 c_3 \tag{6.232}$$

$$c_4 \frac{d}{dt}\theta_4 = \left(\frac{d}{dt}a_x\right)(c_1c_2c_3 - s_1s_3) + a_x\frac{d}{dt}(c_1c_2c_3 - s_1s_3) + \left(\frac{d}{dt}a_y\right)(s_1c_2c_3 + c_1s_3)$$
$$+ a_y\frac{d}{dt}(s_1c_2c_3 + c_1s_3) + \left(\frac{d}{dt}a_z\right)(s_2c_3) + a_z\frac{d}{dt}(s_2c_3) \tag{6.233}$$

To simplify the subsequent derivation, the second, fourth, and sixth terms of Equation (6.233) are evaluated separately. Beginning with the second,

$$a_x\frac{d}{dt}(c_1c_2c_3 - s_1s_3) =$$
$$a_x\left(\frac{d}{dt}(c_1c_2)c_3 + c_1c_2\frac{d}{dt}c_3 - \left(c_1\frac{d}{dt}\theta_1\right)s_3 - s_1\left(c_3\frac{d}{dt}\theta_3\right)\right) \tag{6.234}$$

$$a_x\frac{d}{dt}(c_1c_2c_3 - s_1s_3) = a_x\left(\left(\left(-s_1\frac{d}{dt}\theta_1\right)c_2 + c_1\left(-s_2\frac{d}{dt}\theta_2\right)\right)c_3\right.$$
$$\left. - c_1c_2s_3\frac{d}{dt}\theta_3 - c_1s_3\frac{d}{dt}\theta_1 - s_1c_3\frac{d}{dt}\theta_3\right) \tag{6.235}$$

$$a_x\frac{d}{dt}(c_1c_2c_3 - s_1s_3) =$$
$$-a_x(s_1c_2c_3 + c_1s_3)\frac{d}{dt}\theta_1 - a_xc_1s_2c_3\frac{d}{dt}\theta_2 - a_x(c_1c_2s_3 + s_1c_3)\frac{d}{dt}\theta_3 \tag{6.236}$$

The fourth term of Equation (6.233) evaluates as follows:

$$a_y\frac{d}{dt}(s_1c_2c_3 + c_1s_3) =$$
$$a_y\left(\frac{d}{dt}(s_1c_2)c_3 + s_1c_2\frac{d}{dt}c_3 - s_1\frac{d}{dt}\theta_1s_3 + c_1c_3\frac{d}{dt}\theta_3\right) \tag{6.237}$$

$$a_y\frac{d}{dt}(s_1c_2c_3 + c_1s_3) = a_y\left(\left(\left(c_1\frac{d}{dt}\theta_1\right)c_2 - s_1\left(s_2\frac{d}{dt}\theta_2\right)\right)c_3\right.$$
$$\left. - s_1c_2s_3\frac{d}{dt}\theta_3 - s_1s_3\frac{d}{dt}\theta_1 + c_1c_3\frac{d}{dt}\theta_3\right) \tag{6.238}$$

$$a_y \frac{d}{dt}(s_1 c_2 c_3 + c_1 s_3) = \\ a_y(c_1 c_2 c_3 - s_1 s_3)\frac{d}{dt}\theta_1 - a_y s_1 s_2 c_3 \frac{d}{dt}\theta_2 - a_y(s_1 c_2 s_3 - c_1 c_3)\frac{d}{dt}\theta_3 \tag{6.239}$$

Lastly, the sixth term of Equation (6.233) is expanded.

$$a_z \frac{d}{dt}(s_2 c_3) = a_z\left(\left(c_2 \frac{d}{dt}\theta_2\right)c_3 + s_2\left(-s_3 \frac{d}{dt}\theta_3\right)\right) \tag{6.240}$$

$$a_z \frac{d}{dt}(s_2 c_3) = a_z c_2 c_3 \frac{d}{dt}\theta_2 - a_z s_2 s_3 \frac{d}{dt}\theta_3 \tag{6.241}$$

Equations (6.236), (6.239), and (6.241) are then substituted into Equation (6.233), and the resulting equation is rearranged to reduce the number of operations required for its evaluation.

$$\begin{aligned} c_4 \frac{d}{dt}\theta_4 = {} & \left(\frac{d}{dt}a_x\right)(c_1 c_2 c_3 - s_1 s_3) - a_x(s_1 c_2 c_3 + c_1 s_3)\frac{d}{dt}\theta_1 - a_x c_1 s_2 c_3 \frac{d}{dt}\theta_2 \\ & - a_x(c_1 c_2 s_3 + s_1 c_3)\frac{d}{dt}\theta_3 + \left(\frac{d}{dt}a_y\right)(s_1 c_2 c_3 + c_1 s_3) \\ & + a_y(c_1 c_2 c_3 - s_1 s_3)\frac{d}{dt}\theta_1 - a_y s_1 s_2 c_3 \frac{d}{dt}\theta_2 - a_y(s_1 c_2 s_3 - c_1 c_3)\frac{d}{dt}\theta_3 \\ & + \left(\frac{d}{dt}a_z\right)(s_2 c_3) + a_z c_2 c_3 \frac{d}{dt}\theta_2 - a_z s_2 s_3 \frac{d}{dt}\theta_3 \end{aligned} \tag{6.242}$$

$$\begin{aligned} c_4 \frac{d}{dt}\theta_4 = {} & (c_1 c_2 c_3 - s_1 s_3)\frac{d}{dt}a_x + (s_1 c_2 c_3 + c_1 s_3)\frac{d}{dt}a_y + s_2 c_3 \frac{d}{dt}a_z \\ & + (-a_x(s_1 c_2 c_3 + c_1 s_3) + a_y(c_1 c_2 c_3 - s_1 s_3))\frac{d}{dt}\theta_1 + (-a_x c_1 s_2 c_3 - a_y s_1 s_2 c_3 \\ & + a_z c_2 c_3)\frac{d}{dt}\theta_2 + (-a_x(c_1 c_2 s_3 + s_1 c_3) - a_y(s_1 c_2 s_3 - c_1 c_3) - a_z s_2 s_3)\frac{d}{dt}\theta_3 \end{aligned} \tag{6.243}$$

$$\frac{d}{dt}\theta_4 = \Big((c_1c_2c_3 - s_1s_3)\frac{d}{dt}a_x + (s_1c_2c_3 + c_1s_3)\frac{d}{dt}a_y + s_2c_3\frac{d}{dt}a_z \\ + ((a_yc_1 - a_xs_1)c_2c_3 - (a_xc_1 + a_ys_1)s_3)\frac{d}{dt}\theta_1 + (-(a_xc_1 + a_ys_1)s_2 \\ + a_zc_2)c_3\frac{d}{dt}\theta_2 + (-((a_xc_1 + a_ys_1)c_2 + a_zs_2)s_3 + (a_yc_1 - a_xs_1)c_3)\frac{d}{dt}\theta_3\Big)/c_4 \tag{6.244}$$

Consider the following substitutions for the sake of further reducing the number of operations.

$$A = a_yc_1 - a_xs_1 \tag{6.245}$$

$$B = a_xc_1 + a_ys_1 \tag{6.246}$$

$$\frac{d}{dt}\theta_4 = \Big((c_1c_2c_3 - s_1s_3)\frac{d}{dt}a_x + (s_1c_2c_3 + c_1s_3)\frac{d}{dt}a_y + s_2c_3\frac{d}{dt}a_z \\ + (Ac_2c_3 - Bs_3)\frac{d}{dt}\theta_1 + (-Bs_2 + a_zc_2)c_3\frac{d}{dt}\theta_2 \\ + (-(Bc_2 + a_zs_2)s_3 + Ac_3)\frac{d}{dt}\theta_3\Big)/c_4 \tag{6.247}$$

f. <u>Joint Variable 5</u>. The first of the pair of equations dealing with θ_5 is next differentiated to provide one formula for the rate of θ_5. The process here parallels that just followed for the second θ_4 equation as the original equations take the same form. Beginning with Equation (5.186),

$$c_5 = o_x(c_1c_2s_3 + s_1c_3) + o_y(s_1c_2s_3 - c_1c_3) + o_zs_2s_3 \tag{6.248}$$

$$-s_5\frac{d}{dt}\theta_5 = \left(\frac{d}{dt}o_x\right)(c_1c_2s_3 + s_1c_3) + o_x\frac{d}{dt}(c_1c_2s_3 + s_1c_3) + \left(\frac{d}{dt}o_y\right)(s_1c_2s_3 - c_1c_3) \\ + o_y\frac{d}{dt}(s_1c_2s_3 - c_1c_3) + \left(\frac{d}{dt}o_z\right)(s_2s_3) + o_z\frac{d}{dt}(s_2s_3) \tag{6.249}$$

Expanding term 2,

$$o_x \frac{d}{dt}(c_1c_2s_3 + s_1c_3) = \\ o_x\left(\frac{d}{dt}(c_1c_2)s_3 + c_1c_2\frac{d}{dt}s_3 + \left(c_1\frac{d}{dt}\theta_1\right)c_3 - s_1\left(s_3\frac{d}{dt}\theta_3\right)\right) \tag{6.250}$$

$$o_x \frac{d}{dt}(c_1c_2s_3 + s_1c_3) = o_x\left(\left(\left(-s_1\frac{d}{dt}\theta_1\right)c_2 + c_1\left(-s_2\frac{d}{dt}\theta_2\right)\right)s_3 \\ + c_1c_2c_3\frac{d}{dt}\theta_3 + c_1c_3\frac{d}{dt}\theta_1 - s_1s_3\frac{d}{dt}\theta_3\right) \tag{6.251}$$

$$o_x \frac{d}{dt}(c_1c_2s_3 + s_1c_3) = \\ o_x(-s_1c_2s_3 + c_1c_3)\frac{d}{dt}\theta_1 - o_xc_1s_2s_3\frac{d}{dt}\theta_2 + o_x(c_1c_2c_3 - s_1s_3)\frac{d}{dt}\theta_3 \tag{6.252}$$

Expanding term 4,

$$o_y \frac{d}{dt}(s_1c_2s_3 - c_1c_3) = \\ o_y\left(\frac{d}{dt}(s_1c_2)s_3 + s_1c_2\frac{d}{dt}s_3 + s_1\frac{d}{dt}\theta_1c_3 + c_1s_3\frac{d}{dt}\theta_3\right) \tag{6.253}$$

$$o_y \frac{d}{dt}(s_1c_2s_3 - c_1c_3) = o_y\left(\left(\left(c_1\frac{d}{dt}\theta_1\right)c_2 - s_1\left(s_2\frac{d}{dt}\theta_2\right)\right)s_3 \\ + s_1c_2c_3\frac{d}{dt}\theta_3 + s_1c_3\frac{d}{dt}\theta_1 + c_1s_3\frac{d}{dt}\theta_3\right) \tag{6.254}$$

$$o_y \frac{d}{dt}(s_1c_2s_3 - c_1c_3) = \\ o_y(c_1c_2s_3 + s_1c_3)\frac{d}{dt}\theta_1 - o_ys_1s_2s_3\frac{d}{dt}\theta_2 + o_y(s_1c_2c_3 + c_1s_3)\frac{d}{dt}\theta_3 \tag{6.255}$$

Expanding term 6,

$$o_z \frac{d}{dt}(s_2s_3) = o_z\left(\left(c_2\frac{d}{dt}\theta_2\right)s_3 + s_2\left(c_3\frac{d}{dt}\theta_3\right)\right) \tag{6.256}$$

$$o_z \frac{d}{dt}(s_2 s_3) = o_z c_2 s_3 \frac{d}{dt}\theta_2 + o_z s_2 c_3 \frac{d}{dt}\theta_3 \tag{6.257}$$

Substituting Equations (6.252), (6.255), and (6.257) into Equation (6.249),

$$\begin{aligned}
-s_5 \frac{d}{dt}\theta_5 = {} & \left(\frac{d}{dt}o_x\right)(c_1c_2s_3 + s_1c_3) + o_x(-s_1c_2s_3 + c_1c_3)\frac{d}{dt}\theta_1 \\
& - o_xc_1s_2s_3\frac{d}{dt}\theta_2 + o_x(c_1c_2c_3 - s_1s_3)\frac{d}{dt}\theta_3 + \left(\frac{d}{dt}o_y\right)(s_1c_2s_3 - c_1c_3) \\
& + o_y(c_1c_2s_3 + s_1c_3)\frac{d}{dt}\theta_1 - o_ys_1s_2s_3\frac{d}{dt}\theta_2 + o_y(s_1c_2c_3 + c_1s_3)\frac{d}{dt}\theta_3 \\
& + \left(\frac{d}{dt}o_z\right)(s_2s_3) + o_zc_2s_3\frac{d}{dt}\theta_2 + o_zs_2c_3\frac{d}{dt}\theta_3
\end{aligned} \tag{6.258}$$

$$\begin{aligned}
-s_5 \frac{d}{dt}\theta_5 = {} & (c_1c_2s_3 + s_1c_3)\frac{d}{dt}o_x + (s_1c_2s_3 - c_1c_3)\frac{d}{dt}o_y + s_2s_3\frac{d}{dt}o_z \\
& + (o_x(-s_1c_2s_3 + c_1c_3) + o_y(c_1c_2s_3 + s_1c_3))\frac{d}{dt}\theta_1 + (-o_xc_1s_2s_3 - o_ys_1s_2s_3 \\
& + o_zc_2s_3)\frac{d}{dt}\theta_2 + (o_x(c_1c_2c_3 - s_1s_3) + o_y(s_1c_2c_3 + c_1s_3) + o_zs_2c_3)\frac{d}{dt}\theta_3
\end{aligned} \tag{6.259}$$

$$\begin{aligned}
\frac{d}{dt}\theta_5 = - & \Big((c_1c_2s_3 + s_1c_3)\frac{d}{dt}o_x + (s_1c_2s_3 - c_1c_3)\frac{d}{dt}o_y + s_2s_3\frac{d}{dt}o_z \\
& + ((o_yc_1 - o_xs_1)c_2s_3 + (o_xc_1 + o_ys_1)c_3)\frac{d}{dt}\theta_1 + (-(o_xc_1 + o_ys_1)s_2 \\
& + o_zc_2)s_3\frac{d}{dt}\theta_2 + (((o_xc_1 + o_ys_1)c_2 + o_zs_2)c_3 + (o_yc_1 - o_xs_1)s_3)\frac{d}{dt}\theta_3\Big)/s_5
\end{aligned} \tag{6.260}$$

Substitutions are then made to additionally reduce the number of operations.

$$C = o_yc_1 - o_xs_1 \tag{6.261}$$

$$D = o_xc_1 + o_ys_1 \tag{6.262}$$

$$\frac{d}{dt}\theta_5 = -\bigg((c_1c_2s_3 + s_1c_3)\frac{d}{dt}o_x + (s_1c_2s_3 - c_1c_3)\frac{d}{dt}o_y + s_2s_3\frac{d}{dt}o_z + (Cc_2s_3 + Dc_3)\frac{d}{dt}\theta_1 + (-Ds_2 + o_zc_2)s_3\frac{d}{dt}\theta_2 + ((Dc_2 + o_zs_2)c_3 + Cs_3)\frac{d}{dt}\theta_3\bigg)/s_5 \tag{6.263}$$

Finally, Equation (5.187) provides an alternate solution for the rate of θ_5 in the event that s_5 in the equation above approaches zero.

$$s_5 = n_x(c_1c_2s_3 + s_1c_3) + n_y(s_1c_2s_3 - c_1c_3) + n_zs_2s_3 \tag{6.264}$$

Comparison of this equation with Equation (6.248) will show that the factors of n_x, n_y, and n_z in Equation (6.264) are precisely those of o_x, o_y, and o_z, respectively, in Equation (6.248). Thus the solution obtained by differentiating the cosine of θ_5, given in Equations (6.261) through (6.263), is used as the outline of the solution which would be obtained by differentiating the sine of θ_5 in Equation (6.264). The components of vector $\vec{o}$ are replaced with those of vector $\vec{n}$. Differentiation of the left hand side of Equation (6.264) results in $c_5\frac{d}{dt}\theta_5$ instead of the previous $-s_5\frac{d}{dt}\theta_5$, so the sign of Equation (6.263)'s right hand side is reversed and the denominator becomes c_5.

$$E = n_yc_1 - n_xs_1 \tag{6.265}$$

$$F = n_xc_1 + n_ys_1 \tag{6.266}$$

$$\frac{d}{dt}\theta_5 = \bigg((c_1c_2s_3 + s_1c_3)\frac{d}{dt}n_x + (s_1c_2s_3 - c_1c_3)\frac{d}{dt}n_y + s_2s_3\frac{d}{dt}n_z + (Ec_2s_3 + Fc_3)\frac{d}{dt}\theta_1 + (-Fs_2 + n_zc_2)s_3\frac{d}{dt}\theta_2 + ((Fc_2 + n_zs_2)c_3 + Es_3)\frac{d}{dt}\theta_3\bigg)/c_5 \tag{6.267}$$

2. Numerical Example. The derivative technique for determining joint variable rates shall be demonstrated by using the translational and rotational rates obtained in the first section as input here. The original joint variable rates used as input there should then result as output here.

The original joint variables were stated as follows in Equation (6.73):

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \end{bmatrix} = \begin{bmatrix} -115^\circ \\ 25^\circ \\ 50^\circ \\ 65^\circ \\ -35^\circ \end{bmatrix} \tag{6.268}$$

Recall also from Chapter 5's Equation (5.15) the corresponding position and orientation matrix.

$$\begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.790 & -0.516 & 0.331 & 39.566 \\ 0.195 & -0.300 & -0.934 & -260.692 \\ 0.581 & 0.802 & -0.137 & 55.745 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.269}$$

The results obtained from the forward application of the Jacobian in Equation (6.87) were as follows:

$$\frac{d}{dt}\begin{bmatrix} d_x \\ d_y \\ d_z \\ \delta_x \\ \delta_y \\ \delta_z \end{bmatrix} = {}^{0}\begin{bmatrix} 196.548 \\ 64.286 \\ -75.924 \\ 0.107 \\ 0.341 \\ 1.085 \end{bmatrix} \tag{6.270}$$

The rates of change of the components of the orientation vectors are obtained from the matrix multiplication of Equation (6.196).

$$\frac{d}{dt}\begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} = \begin{bmatrix} 0 & -{}^{0}\delta_z & {}^{0}\delta_y \\ {}^{0}\delta_z & 0 & -{}^{0}\delta_x \\ -{}^{0}\delta_y & {}^{0}\delta_x & 0 \end{bmatrix}\begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} \tag{6.271}$$

$$\frac{d}{dt}\begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} = \begin{bmatrix} 0 & -1.085 & 0.341 \\ 1.085 & 0 & -0.107 \\ -0.341 & 0.107 & 0 \end{bmatrix}\begin{bmatrix} 0.790 & -0.516 & 0.331 \\ 0.195 & -0.300 & -0.934 \\ 0.581 & 0.802 & -0.137 \end{bmatrix} \tag{6.272}$$

$$\frac{d}{dt}\begin{bmatrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{bmatrix} = \begin{bmatrix} -0.013 & 0.599 & 0.967 \\ 0.795 & -0.646 & 0.374 \\ -0.249 & 0.144 & -0.213 \end{bmatrix} \tag{6.273}$$

Next, the coordinates of the end of the arm proper, p_{a_x}, p_{a_y}, and p_{a_z}, are determined from Equations (6.210), (6.212), and (6.214) for use in determining the rates of the three arm joint variables.

$$p_{a_x} = p_x - d_5 a_x \tag{6.274}$$

$$p_{a_x} = 39.566 - 100(0.331) = 6.466 \tag{6.275}$$

$$p_{a_y} = p_y - d_5 a_y \tag{6.276}$$

$$p_{a_y} = -260.692 - 100(-0.934) = -167.292 \tag{6.277}$$

$$p_{a_z} = p_z - d_5 a_z \tag{6.278}$$

$$p_{a_z} = 55.745 - 100(-0.137) = 69.445 \tag{6.279}$$

The rates of change for the coordinates for the end of the arm proper, $\frac{d}{dt} p_{a_x}$, $\frac{d}{dt} p_{a_y}$, and $\frac{d}{dt} p_{a_z}$, are determined next from Equations (6.211), (6.213), and (6.215), respectively.

$$\frac{d}{dt} p_{a_x} = \frac{d}{dt} p_x - d_5 \frac{d}{dt} a_x \tag{6.280}$$

$$\frac{d}{dt} p_{a_x} = 196.548 - 100(0.967) = 99.848 \tag{6.281}$$

$$\frac{d}{dt} p_{a_y} = \frac{d}{dt} p_y - d_5 \frac{d}{dt} a_y \tag{6.282}$$

$$\frac{d}{dt} p_{a_y} = 64.287 - 100(0.374) = 26.887 \tag{6.283}$$

$$\frac{d}{dt} p_{a_z} = \frac{d}{dt} p_z - d_5 \frac{d}{dt} a_z \tag{6.284}$$

$$\frac{d}{dt} p_{a_z} = -75.919 - 100(-0.213) = -54.619 \tag{6.285}$$

The joint rates are then determined by substitution into the equations derived for each. The rate for θ_3 was obtained first in Equation (6.203).

$$\frac{d}{dt}\theta_3 = \frac{p_{a_x}\frac{d}{dt}p_{a_x} + p_{a_y}\frac{d}{dt}p_{a_y} + p_{a_z}\frac{d}{dt}p_{a_z}}{-a_2a_3s_3} \tag{6.286}$$

$$\frac{d}{dt}\theta_3 = \frac{6.466(99.848) - 167.292(26.887) + 69.445(-54.619)}{-100(100)\sin(50^\circ)} = 0.998 \tag{6.287}$$

The original rate selected in the first section for θ_3 and given by Equation (6.86) was 1.0.

The rate for θ_1 was next provided by Equation (6.219).

$$\frac{d}{dt}\theta_1 = \frac{c_1\frac{d}{dt}p_{a_y} - s_1\frac{d}{dt}p_{a_x} - a_3c_3\frac{d}{dt}\theta_3}{p_{a_x}c_1 + p_{a_y}s_1} \tag{6.288}$$

$$\frac{d}{dt}\theta_1 = \frac{\cos(-115^\circ)(26.887) - \sin(-115^\circ)(99.818) - 100\cos(50^\circ)(0.998)}{6.466\cos(-115^\circ) - 167.292\sin(-115^\circ)} \tag{6.289}$$

$$\frac{d}{dt}\theta_1 = 0.100 \tag{6.290}$$

The original rate selected for θ_1 was 0.1

Two expressions were obtained for the rate of θ_2. The first was Equation (6.223).

$$\frac{d}{dt}\theta_2 = \frac{(c_1p_{a_y} - s_1p_{a_x})\frac{d}{dt}\theta_1 + c_1\frac{d}{dt}p_{a_x} + s_1\frac{d}{dt}p_{a_y} + a_3c_2s_3\frac{d}{dt}\theta_3}{-s_2(a_3c_3 + a_2)} \tag{6.291}$$

$$\begin{aligned}\frac{d}{dt}\theta_2 = {} & ((\cos(-115^\circ)(-167.292) - \sin(-115^\circ)(6.466))(0.100) \\ & + \cos(-115^\circ)(99.818) + \sin(-115^\circ)(26.887) \\ & + 100\cos 25^\circ \sin 50^\circ(0.998))/((-\sin 25^\circ)(100\cos 50^\circ + 100))\end{aligned} \tag{6.292}$$

$$\frac{d}{dt}\theta_2 = -0.150 \tag{6.293}$$

The second expression for the rate of θ_2 was given by Equation (6.226).

$$\frac{d}{dt}\theta_2 = \frac{\frac{d}{dt}p_{a_z} + a_3 s_2 s_3 \frac{d}{dt}\theta_3}{c_2(a_3 c_3 + a_2)} \tag{6.294}$$

$$\frac{d}{dt}\theta_2 = \frac{-54.619 + 100 \sin 25^\circ \sin 50^\circ (0.998)}{\cos 25^\circ (100 \cos 50^\circ + 100)} = -0.150 \tag{6.295}$$

As can be seen, the two results are identical; further, the original θ_2 rate was selected as -0.150.

Each of the wrist variables also has two solutions. The first for θ_4 was given in Equation (6.231).

$$\begin{aligned}\frac{d}{dt}\theta_4 = \Big(&(a_x s_1 - a_y c_1) s_2 \frac{d}{dt}\theta_1 - ((a_x c_1 + a_y s_1) c_2 + a_z s_2) \frac{d}{dt}\theta_2 \\ &- c_1 s_2 \frac{d}{dt} a_x - s_1 s_2 \frac{d}{dt} a_y + c_2 \frac{d}{dt} a_z \Big) / s_4\end{aligned} \tag{6.296}$$

$$\begin{aligned}\frac{d}{dt}\theta_4 = &((0.331 \sin(-115^\circ) - (-0.934)\cos(-115^\circ)) \sin 25^\circ (0.100) \\ &- ((0.331 \cos(-115^\circ) + (-0.934)\sin(-115^\circ)) \cos 25^\circ \\ &+ (-0.137) \sin 25^\circ)(-0.150) - \cos(-115^\circ) \sin 25^\circ (0.967) \\ &- \sin(-115^\circ) \sin 25^\circ (0.374) + \cos 25^\circ (-0.213)) / \sin 65^\circ\end{aligned} \tag{6.297}$$

$$\frac{d}{dt}\theta_4 = 0.200 \tag{6.298}$$

The second solution for θ_4 was given by Equations (6.245), (6.246), and (6.247).

$$A = a_y c_1 - a_x s_1 \tag{6.299}$$

$$B = a_x c_1 + a_y s_1 \tag{6.300}$$

$$\begin{aligned} \frac{d}{dt}\theta_4 = \Big(& (c_1c_2c_3 - s_1s_3)\frac{d}{dt}a_x + (s_1c_2c_3 + c_1s_3)\frac{d}{dt}a_y + s_2c_3\frac{d}{dt}a_z \\ & + (Ac_2c_3 - Bs_3)\frac{d}{dt}\theta_1 + (-Bs_2 + a_zc_2)c_3\frac{d}{dt}\theta_2 \\ & + (-(Bc_2 + a_zs_2)s_3 + Ac_3)\frac{d}{dt}\theta_3\Big)/c_4 \end{aligned} \tag{6.301}$$

$$A = -0.934\cos(-115^\circ) - 0.331\sin(-115^\circ) = 0.695 \tag{6.302}$$

$$B = 0.331\cos(-115^\circ) + (-0.934)\sin(-115^\circ) = 0.707 \tag{6.303}$$

$$\begin{aligned} \frac{d}{dt}\theta_4 = ((& \cos(-115^\circ)\cos 25^\circ\cos 50^\circ - \sin(-115^\circ)\sin 50^\circ)(0.967) \\ & + (\sin(-115^\circ)\cos 25^\circ\cos 50^\circ + \cos(-115^\circ)\sin 50^\circ)(0.374) \\ & + \sin 25^\circ\cos 50^\circ(-0.213) \\ & + (0.695\cos 25^\circ\cos 50^\circ - 0.707\sin 50^\circ)(0.100) \\ & + (-0.707\sin 25^\circ + (-0.137)\cos 25^\circ)\cos 50^\circ(-0.150) \\ & + (-(0.707\cos 25^\circ + (-0.137)\sin 25^\circ)\sin 50^\circ \\ & \quad + 0.695\cos 50^\circ)(0.998))/\cos 65^\circ \end{aligned} \tag{6.304}$$

$$\frac{d}{dt}\theta_4 = 0.199 \tag{6.305}$$

The two rates for θ_4, 0.200 and 0.199, are very near one another; the original θ_4 rate was 0.2.

Finally, two solutions exist for the rate of θ_5, the first of which was given by Equations (6.261), (6.262), and (6.263).

$$C = o_y c_1 - o_x s_1 \tag{6.306}$$

$$D = o_x c_1 + o_y s_1 \tag{6.307}$$

$$\begin{aligned}\frac{d}{dt}\theta_5 = -\Big((c_1c_2s_3 + s_1c_3)\frac{d}{dt}o_x + (s_1c_2s_3 - c_1c_3)\frac{d}{dt}o_y + s_2s_3\frac{d}{dt}o_z \\ + (Cc_2s_3 + Dc_3)\frac{d}{dt}\theta_1 + (-Ds_2 + o_zc_2)s_3\frac{d}{dt}\theta_2 \\ + ((Dc_2 + o_zs_2)c_3 + Cs_3)\frac{d}{dt}\theta_3\Big)/s_5\end{aligned} \tag{6.308}$$

$$C = -0.300\cos(-115^\circ) - (-0.516)\sin(-115^\circ) = -0.341 \tag{6.309}$$

$$D = -0.516\cos(-115^\circ) + (-0.300)\sin(-115^\circ) = 0.490 \tag{6.310}$$

$$\begin{aligned}\frac{d}{dt}\theta_5 = -((\cos(-115^\circ)\cos 25^\circ\sin 50^\circ + \sin(-115^\circ)\cos 50^\circ)(0.599) \\ + (\sin(-115^\circ)\cos 25^\circ\sin 50^\circ - \cos(-115^\circ)\cos 50^\circ)(-0.646) \\ + \sin 25^\circ\sin 50^\circ(0.144) \\ + (-0.341\cos 25^\circ\sin 50^\circ + 0.490\cos 50^\circ)(0.100) \\ + (-0.490\sin 25^\circ + 0.802\cos 25^\circ)\sin 50^\circ(-0.150) \\ + ((0.490\cos 25^\circ + 0.802\sin 25^\circ)\cos 50^\circ \\ + (-0.341)\sin 50^\circ)(0.998))/\sin(-35^\circ)\end{aligned} \tag{6.311}$$

$$\frac{d}{dt}\theta_5 = -0.100 \tag{6.312}$$

The alternate solution for the rate of θ_5 was given by Equations (6.265), (6.266), and (6.267).

$$E = n_yc_1 - n_xs_1 \tag{6.313}$$

$$F = n_xc_1 + n_ys_1 \tag{6.314}$$

$$\frac{d}{dt}\theta_5 = \Big((c_1c_2s_3 + s_1c_3)\frac{d}{dt}n_x + (s_1c_2s_3 - c_1c_3)\frac{d}{dt}n_y + s_2s_3\frac{d}{dt}n_z$$
$$+ (Ec_2s_3 + Fc_3)\frac{d}{dt}\theta_1 + (-Fs_2 + n_zc_2)s_3\frac{d}{dt}\theta_2 \qquad (6.315)$$
$$+ ((Fc_2 + n_zs_2)c_3 + Es_3)\frac{d}{dt}\theta_3\Big)/c_5$$

$$E = 0.195\cos(-115^\circ) - 0.790\sin(-115^\circ) = 0.634 \qquad (6.316)$$

$$F = 0.790\cos(-115^\circ) + 0.195\sin(-115^\circ) = -0.511 \qquad (6.317)$$

$$\frac{d}{dt}\theta_5 = ((\cos(-115^\circ)\cos 25^\circ\sin 50^\circ + \sin(-115^\circ)\cos 50^\circ)(-0.013)$$
$$+ (\sin(-115^\circ)\cos 25^\circ\sin 50^\circ - \cos(-115^\circ)\cos 50^\circ)(0.795)$$
$$+ \sin 25^\circ\sin 50^\circ(-0.249)$$
$$+ (0.634\cos 25^\circ\sin 50^\circ + (-0.511)\cos 50^\circ)(0.100) \qquad (6.318)$$
$$+ (-(-0.511)\sin 25^\circ + 0.581\cos 25^\circ)\sin 50^\circ(-0.150)$$
$$+ ((-0.511\cos 25^\circ + 0.581\sin 25^\circ)\cos 50^\circ$$
$$+ 0.634\sin 50^\circ)(0.998))/\cos(-35^\circ)$$

$$\frac{d}{dt}\theta_5 = -0.101 \qquad (6.319)$$

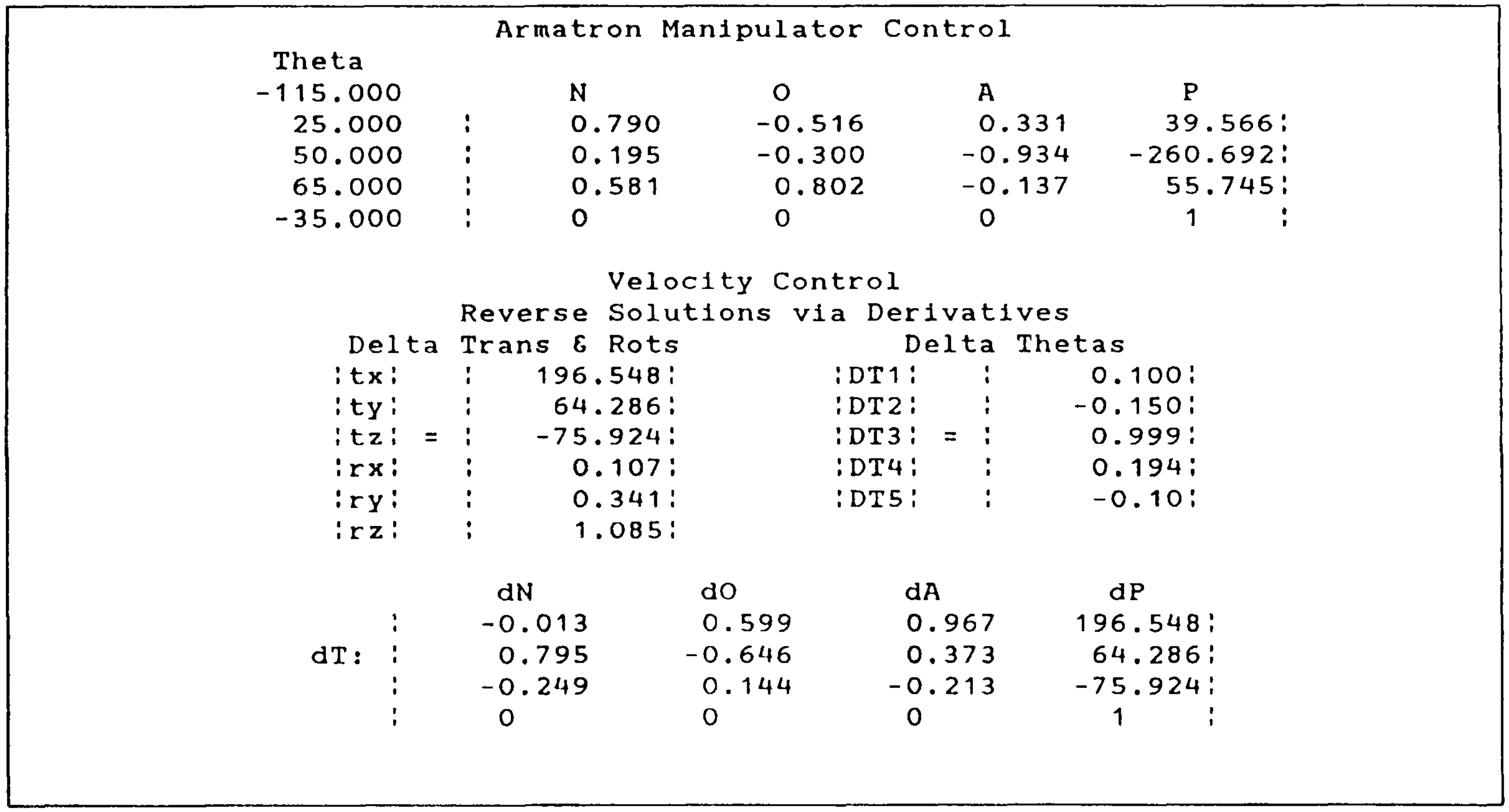
The two values for θ_5, -0.100 and -0.101, are close to one another; the original rate selected for θ_5 was -0.1.

3. Program Control. The procedure for the solution of joint rates base on the differentiation of control variable equations is named **rev_sol_via_deriv**. The body of the procedure follows:

```
sin_cos (theta, s, c);
dsply_rsvd (&mr, mc, vr, vc);
do
   {
   get_delta_trans_rot (delta_trans_rot, vr[0], vc[0]);
   calc_delta_noap (noap, delta_trans_rot, dnoap, mc, mr);
   calc_delta_theta (s, c, dnoap, noap, dtheta, vr[1], vc[1]);
   query_ch = cont ("different rates");
}
while ( query_ch == 'Y' );
```

After obtaining all six translational and rotational rates, the delta T or delta n-o-a-p matrix is calculated. Subsequent to that, the joint rates are obtained and the user is prompted as to whether to continue under this topic. As was the case for the forward solution of translational and rotational rates via the Jacobian, the procedures involved are straightforward in design and need not be discussed further here.

4. Program Example. Figure 6.18 depicts the single display associated with this solution technique. The delta T matrix shown may be compared with Equation 6.273 to see that only minor differences exist. The joint variable rates show were arrived at in Equations (6.287), (6.289), (6.293) and (6.295), (6.298) and (6.305), and (6.312) and (6.319) in the order theta 3, 1, 2, 4, and 5.



```
                    Armatron Manipulator Control
      Theta
     -115.000            N          O          A          P
       25.000    :     0.790     -0.516      0.331     39.566:
       50.000    :     0.195     -0.300     -0.934   -260.692:
       65.000    :     0.581      0.802     -0.137     55.745:
      -35.000    :     0 	         0          0          1     :

                          Velocity Control
                   Reverse Solutions via Derivatives
         Delta Trans & Rots                 Delta Thetas
         :tx:     :   196.548:            :DT1:    :      0.100:
         :ty:     :    64.286:            :DT2:    :     -0.150:
         :tz:  =  :   -75.924:            :DT3: =  :      0.999:
         :rx:     :     0.107:            :DT4:    :      0.194:
         :ry:     :     0.341:            :DT5:    :     -0.10:
         :rz:     :     1.085:

                     dN          dO          dA          dP
               :   -0.013       0.599       0.967     196.548:
         dT:   :    0.795      -0.646       0.373      64.286:
               :   -0.249       0.144      -0.213     -75.924:
               :    0           0           0           1     :
```

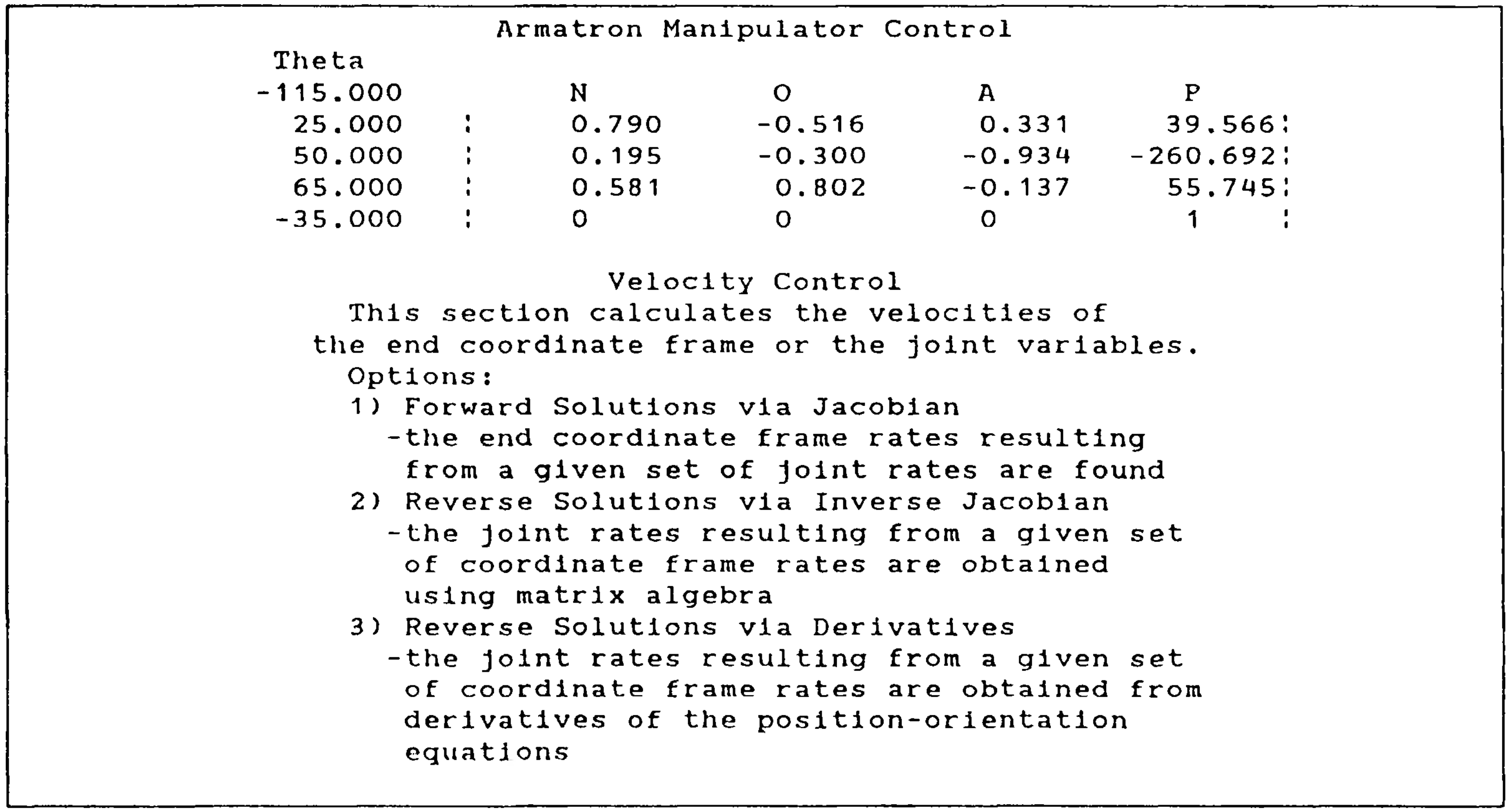
Figure 6.18. Reverse Solutions via Derivatives Display

D. THE CONTROLLING PROCEDURE

Procedure `velocity_control` begins by initializing the joint variables, joint, translational, and rotational rates, and position and orientation matrix. The rates are set to zero while the joint variables and positin and orientation matrix are copied from the current values; this is done so that the originals will remain unaffected as no movement is performed by this chapter. The procedure continues by displaying first an introductory screen, depicted in Figure 6.19, and then an options screen, shown in Figure 6.20

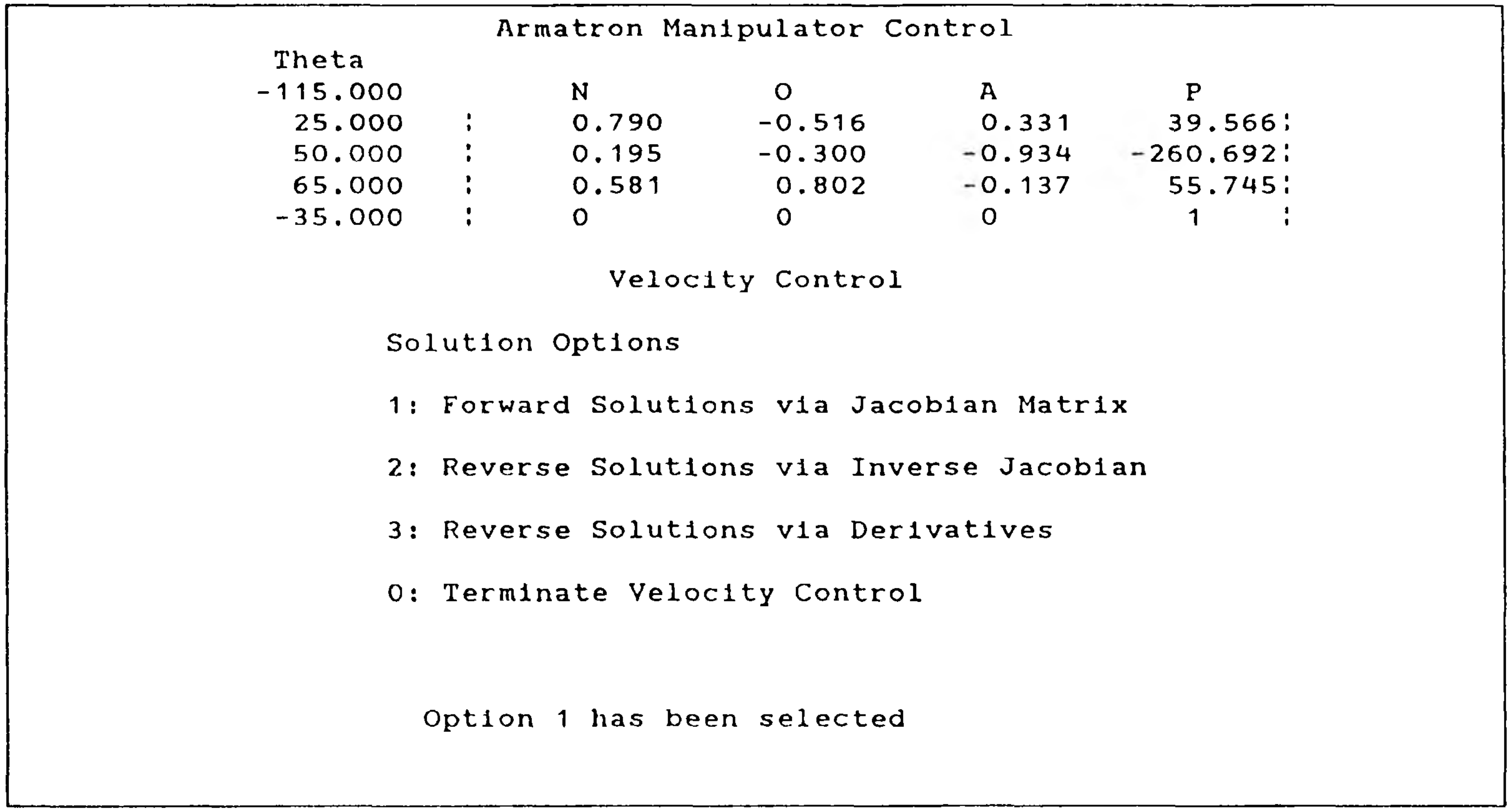
```
for (i = 1; i <= 5; i++)
   {
   theta[i] = original_theta[i];
   dtheta[i] = 0;
   }
for (i = 1; i <= 6; i++)
   delta_trans_rot[i] = 0;
for (i = 0; i <= 3; i++)
   for (j = 0; j <= 2; j++)
      noap[i][j] = original_noap[i][j];

dsply_velocity_introduction ( );
wait_then_erase (9);
dsply_vc_selection ( );
while ((opt = get_option(3)) != 0)
   {
   prompt_msg1 = "Enter New Theta Values? (Y/N)";
   prompt_msg2 = "(<N> = continue with previous values)";
   qc = prompt_input_char (prompt_msg1, prompt_msg2);
   if (qc == 'Y')
      {
      get_theta (theta, row, cols[4]);
      noap_matrix (theta, noap, row, cols);
      }
   wait_then_erase (9);
   switch (opt)
      {
      case 1 : for_sol_via_jac (theta, dtheta);
               break;
      case 2 : rev_sol_via_ij  (theta, delta_trans_rot);
               break;
      case 3 : rev_sol_via_deriv (theta, noap, delta_trans_rot);
               break;
      }
   dsply_vc_selection ( );
   }
wait_then_erase (8);
noap_matrix (original_theta, original_noap, row, cols);
```

The procedure then iterates while velocity control options are selected. After an option has been chosen, the opportunity is presented to change the settings of the joint variables; these values cannot be changed within an option. Velocity control is terminated when an option of 0 is selected. The documented listing for the procedures associated with the velocity control portion of the overall program may be found in Appendix F.



```
                    Armatron Manipulator Control
     Theta
    -115.000          N          O          A          P
      25.000   :    0.790     -0.516      0.331     39.566:
      50.000   :    0.195     -0.300     -0.934   -260.692:
      65.000   :    0.581      0.802     -0.137     55.745:
     -35.000   :    0                 0                 0              1     :

                       Velocity Control
        This section calculates the velocities of
      the end coordinate frame or the joint variables.
        Options:
        1) Forward Solutions via Jacobian
          -the end coordinate frame rates resulting
           from a given set of joint rates are found
        2) Reverse Solutions via Inverse Jacobian
          -the joint rates resulting from a given set
           of coordinate frame rates are obtained
           using matrix algebra
        3) Reverse Solutions via Derivatives
          -the joint rates resulting from a given set
           of coordinate frame rates are obtained from
           derivatives of the position-orientation
           equations
```

Figure 6.19. Velocity Control Introductory Display



```
                   Armatron Manipulator Control
       Theta
      -115.000          N          O          A          P
        25.000   :    0.790     -0.516      0.331     39.566:
        50.000   :    0.195     -0.300     -0.934   -260.692:
        65.000   :    0.581      0.802     -0.137     55.745:
       -35.000   :    0              0          0          1     :

                         Velocity Control

               Solution Options

               1: Forward Solutions via Jacobian Matrix

               2: Reverse Solutions via Inverse Jacobian

               3: Reverse Solutions via Derivatives

               0: Terminate Velocity Control


                 Option 1 has been selected
```

Figure 6.20. Velocity Control Menu Display

VII. TRAJECTORY CONTROL

The topic of this chapter, controlling the trajectory of the manipulator end, is one that is not as heavily manipulator dependent as the topics of previous chapters. Thus the Armatron configuration will not play as significant a role as before. However, trajectory control is fundamental in robotics, and as such it is included here. Another interesting aspect of trajectory control is its strong connection to computer graphics in its use of spline polynomials; see [Fole83] for further comparisons.

A manipulator trajectory can be defined in either of two ways. The first type of trajectory consists of the three dimensional coordinates representing points or nodes through which it is desired to have the manipulator end frame origin pass. Associated with each of these nodes will be a desired orientation of the manipulator end coordinate frame. The second type of trajectory is a sequence of joint variable combinations which the joints are to attain. This is essentially the set of solutions of the nodal positions and orientations of the first trajectory type. The process derived in this chapter shall develop this second type of trajectory; the material in Chapter 5 would allow extension to the first type. The sets of joint variable values to be attained along the trajectory may be thought of as points or nodes just as the coordinates of the trajectory of the first type are; the joint variable sets will be referred to as points or nodes throughout this chapter. The derivations will develop third- and fourth-order polynomials which meet the necessary conditions. While the process is carried out in the geometric sense, it applies equally in terms of joint variables. The development of the program is carried out in stages, in parallel with the derivation of equations and numerical example.

A. SPLINE POLYNOMIALS

Equations shall be developed in this section to define the settings of the joint control variables at any time throughout the traversal of the trajectory; more specifically, individual equations shall be generated to define the value of each joint variable individually with respect to a common time t. In addition to passing through each of the path nodes, the equations developed will also provide continuity in velocity and acceleration through each of the nodes.

1. Derivation of Equations.

The derivation of this section shall develop trajectory equations for one of the manipulator's five joints. The problem then becomes one of fitting curves to a set of points in a plane.

a. Distance-Based Time Units.

The problem to be solved has only one parameter as presented, and that is the nodal values which the joint variables must attain. Nothing is required of the manipulator as to how quickly the path is to be traversed along any given segment. An amount of time measured in as yet unknown units shall first be allocated for the traversal of the trajectory between any two path nodes. Consider an arbitrary pair of successive nodes along the trajectory, i and $i+1$, as depicted in Figure 7.1. While the range different joints have to cover between nodes i and $i+1$ may differ, the amount of time to be used by each does not. Each of the Armatron's five joints must move the difference between its value at node i and $i+1$ in the same amount of time. An arbitrary amount of time shall be allocated based upon the "distance" between successive nodes for use by each of the five joints; the time between nodes i and $i+1$ shall be defined as t_i.

$$t_i = \sqrt{\sum_{j=1}^{5}(\theta_{j_{i+1}} - \theta_{j_i})^2} \tag{7.1}$$

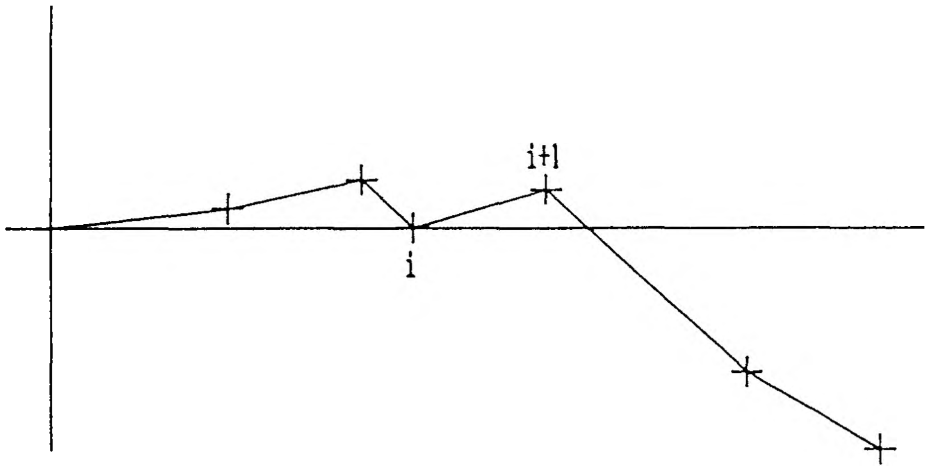


Figure 7.1. Trajectory for a Single Joint Variable

The equations in the following sections shall be derived with respect to the arbitrary units of time defined by Equation (7.1). Subsequent to that, a scaling factor will be developed for traversal of the trajectory in a minimum of time.

b. Internal Spline Segments. Consider again the arbitrary internal segment along the path depicted in Figure 7.1 from point P_i to point P_{i+1}. The spline polynomial for this segment, S_i, will have four boundary conditions placed on it, the first two of which state that the polynomial pass through the segment endpoints.

$$S_i(0) = P_i \tag{7.2}$$

$$S_i(t_i) = P_{i+1} \tag{7.3}$$

The second pair of conditions states that the velocity of the polynomial take on specific, but as yet undetermined, values.

$$S'_i(0) = P'_i \tag{7.4}$$

$$S'_i(t_i) = P'_{i+1} \tag{7.5}$$

The velocities at the end and beginning of adjacent segments will eventually be made equal; this fact does not play a part in the development of the spline polynomial at this point, however.

In order to meet the four specified conditions, the spline polynomial must have four coefficients, or be cubic.

$$S_i(t) = A_i + B_i t + C_i t^2 + D_i t^3 \tag{7.6}$$

The coefficients of $S_i(t)$ are found by using the boundary conditions above. The endpoint conditions for the segment are substituted into Equation (7.6) first.

$$S_i(0) = A_i + B_i(0) + C_i(0)^2 + D_i(0)^3 \tag{7.7}$$

$$P_i = A_i \tag{7.8}$$

$$S_i(t_i) = A_i + B_i(t_i) + C_i(t_i)^2 + D_i(t_i)^3 \tag{7.9}$$

$$P_{i+1} = A_i + B_i t_i + C_i t_i^2 + D_i t_i^3 \tag{7.10}$$

The spline polynomial is then differentiated so that the velocity conditions may be used.

$$S'_i(t) = B_i + 2C_i t + 3D_i t_2 \tag{7.11}$$

$$S'_i(0) = B_i + 2C_i(0) + 3D_i(0)^2 \tag{7.12}$$

$$P'_i = B_i \tag{7.13}$$

$$S'_i(t_i) = B_i + 2C_i(t_i) + 3D_i(t_i)^2 \tag{7.14}$$

$$P'_{i+1} = B_i + 2C_i t_i + 3D_i t_i^2 \tag{7.15}$$

Coefficients A_i and B_i are defined explicitly by Equations (7.8) and (7.13), respectively, while coefficients C_i and D_i are determined from manipulating Equations (7.10) and (7.15) after A_i and B_i have been removed by substitution. First, C_i is obtained by multiplying Equation (7.10) by 3, multiplying Equation (7.15) by $-t_i$, and adding the results.

$$3P_{i+1} = 3P_i + 3P'_i t_i + 3C_i t_i^2 + 3D_i t_i^3 \tag{7.16}$$

$$-P'_{i+1} t_i = \qquad - P'_i t_i - 2C_i t_i^2 - 3D_i t_i^3 \tag{7.17}$$

$$3P_{i+1} - P'_{i+1} t_i = 3P_i + 2P'_i t_i + C_i t_i^2 \tag{7.18}$$

$$C_i = 3\frac{P_{i+1}}{t_i^2} - \frac{P'_{i+1}}{t_i} - 3\frac{P_i}{t_i^2} - 2\frac{P'_i}{t_i} \tag{7.19}$$

$$C_i = 3\frac{(P_{i+1} - P_i)}{t_i^2} - \frac{(P'_{i+1} + 2P'_i)}{t_i} \tag{7.20}$$

Coefficient D_i is next obtained by multiplying Equation (7.10) by 2, multiplying Equation (7.15) by $-t_i$, and adding the results.

$$2P_{i+1} = 2P_i + 2P'_i t_i + 2C_i t_i^2 + 2D_i t_i^3 \tag{7.21}$$

$$-P'_{i+1} t_i = \qquad -P'_i t_i - 2C_i t_i^2 - 3D_i t_i^3 \tag{7.22}$$

$$2P_{i+1} - P'_{i+1} t_i = 2P_i + P'_i t_i - D_i t_i^3 \tag{7.23}$$

$$D_i = \frac{2P_i}{t_i^3} + \frac{P'_i}{t_i^2} - \frac{2P_{i+1}}{t_i^3} + \frac{P'_{i+1}}{t_i^2} \tag{7.24}$$

$$D_i = 2\frac{(P_i - P_{i+1})}{t_i^3} + \frac{(P'_i + P'_{i+1})}{t_i^2} \tag{7.25}$$

The equations for the four coefficients apply for $2 \leq i \leq n-2$, that is, each of the spline segments except the first, 1, and last, n-1. Additional conditions will be seen to exist on these terminal segments.

The coefficient equations depend upon the length of time spent on the segment interval, the segment endpoints, and the velocities at the segment endpoints. These endpoint velocities shall now be defined so that the acceleration at the end of one segment is equal to the acceleration at the beginning of the next. This will guarantee a smooth velocity at each path node. This is accomplished by equating acceleration equations, as they have not been employed yet.

The first derivative, or velocity function, of the segment polynomial in Equation (7.11) is differentiated again to produce the segment's acceleration function.

$$S''_i(t) = 2C_i + 6D_i t \tag{7.26}$$

First consider the acceleration at the end of the i^{th} segment; Equations (7.20) and (7.25) supply substitutions for C_i and D_i, respectively.

$$S''_i(t_i) = 2C_i + 6D_i t_i \tag{7.27}$$

$$S''_i(t_i) = 2\left[3\frac{(P_{i+1} - P_i)}{t_i^2} - \frac{(P'_{i+1} + 2P'_i)}{t_i}\right] + 6\left[2\frac{(P_i - P_{i+1})}{t_i^3} + \frac{(P'_i + P'_{i+1})}{t_i^2}\right]t_i \tag{7.28}$$

$$S''_i(t_i) = 6\frac{(P_{i+1} - P_i)}{t_i^2} - 2\frac{(P'_{i+1} + 2P'_i)}{t_i} + 12\frac{(P_i - P_{i+1})}{t_i^2} + 6\frac{(P'_i + P'_{i+1})}{t_i} \tag{7.29}$$

$$S''_i(t_i) = 6\frac{(P_i - P_{i+1})}{t_i^2} + 2\frac{(P'_i + 2P'_{i+1})}{t_i} \tag{7.30}$$

Next, consider the acceleration at the beginning of segment $i+1$. Coefficient C_{i+1} is substituted for by Equation (7.20).

$$S''_{i+1}(0) = 2C_{i+1} + 6D_{i+1}(0) \tag{7.31}$$

$$S''_{i+1}(0) = 2\left[3\frac{(P_{i+2} - P_{i+1})}{t_{i+1}^2} - \frac{(P'_{i+2} + 2P'_{i+1})}{t_{i+1}}\right] \tag{7.32}$$

The segment i end accelerations in Equations (7.30) and (7.32) are then set equal.

$$6\frac{(P_i - P_{i+1})}{t_i^2} + 2\frac{(P'_i + 2P'_{i+1})}{t_i} = 2\left[3\frac{(P_{i+2} - P_{i+1})}{t_{i+1}^2} - \frac{(P'_{i+2} + 2P'_{i+1})}{t_{i+1}}\right] \tag{7.33}$$

$$3\frac{(P_i - P_{i+1})}{t_i^2} + \frac{(P'_i + 2P'_{i+1})}{t_i} = 3\frac{(P_{i+2} - P_{i+1})}{t_{i+1}^2} - \frac{(P'_{i+2} + 2P'_{i+1})}{t_{i+1}} \tag{7.34}$$

$$\frac{(P'_{i+2} + 2P'_{i+1})}{t_{i+1}} + \frac{(P'_i + 2P'_{i+1})}{t_i} = 3\frac{(P_{i+2} - P_{i+1})}{t_{i+1}^2} - 3\frac{(P_i - P_{i+1})}{t_i^2} \tag{7.35}$$

$$P'_{i+2} + 2P'_{i+1} + (P'_i + 2P'_{i+1})\frac{t_{i+1}}{t_i} = 3\frac{(P_{i+2} - P_{i+1})}{t_{i+1}} - 3(P_i - P_{i+1})\frac{t_{i+1}}{t_i^2} \tag{7.36}$$

$$(P'_{i+2} + 2P'_{i+1})t_i + (P'_i + 2P'_{i+1})t_{i+1} = 3(P_{i+2} - P_{i+1})\frac{t_i}{t_{i+1}} - 3(P_i - P_{i+1})\frac{t_{i+1}}{t_i} \tag{7.37}$$

$$t_iP'_{i+2} + 2t_iP'_{i+1} + t_{i+1}P'_i + 2t_{i+1}P'_{i+1} = 3(P_{i+2} - P_{i+1})\frac{t_i}{t_{i+1}} - 3(P_i - P_{i+1})\frac{t_{i+1}}{t_i} \tag{7.38}$$

$$t_i P'_{i+2} + 2(t_i + t_{i+1})P'_{i+1} + t_{i+1}P'_i = 3(P_{i+2} - P_{i+1})\frac{t_i}{t_{i+1}} + 3(P_{i+1} - P_i)\frac{t_{i+1}}{t_i} \qquad (7.39)$$

Note that this equation is defined for pairs of spline segments i and $i + 1$. The first pair for which the equation holds is $i = 2$ and $i + 1 = 3$ as spline 2 is the first cubic polynomial. The last pair for which the equation holds is $i = n - 3$ and $i + 1 = n - 2$, since spline $n - 2$ is the last cubic polynomial. The index i for the equation then ranges from 2 to $n - 3$. This is a total of $n - 4$ equations, but there are $n - 2$ unknown velocities at the internal trajectory nodes. When the spline polynomials, and hence the acceleration functions, for the first and last trajectory segments are defined, two more velocity equations will result and all of the internal node velocities may be determined.

c. The First Spline Segment. The first spline segment differs from the intermediates in that it has five, instead of four, initial conditions to satisfy. Like the intermediate segments, the polynomial must pass through its endpoints.

$$S_1(0) = P_1 \qquad (7.40)$$

$$S_1(t_1) = P_2 \qquad (7.41)$$

The velocity at the start of this first segment is zero as the manipulator has not begun to move, while the velocity at the endpoint P_2 takes on an as yet undetermined value.

$$S'_1(0) = 0 \qquad (7.42)$$

$$S'_1(t_1) = P'_2 \qquad (7.43)$$

In addition to these constraints, the acceleration at the start of the segment must also be zero, again reflecting the motionless state of the manipulator.

$$S''_1(0) = 0 \tag{7.44}$$

Five coefficients are required of a polynomial to meet these conditions, so it must be a quartic.

$$S_1(t) = A_1 + B_1 t + C_1 t^2 + D_1 t^3 + E_1 t^4 \tag{7.45}$$

The coefficients of $S_1(t)$ are obtained by substitutions from the boundary conditions, beginning with the endpoint conditions.

$$S_1(0) = A_1 + B_1(0) + C_1(0)^2 + D_1(0)^3 + E_1(0)^4 \tag{7.46}$$

$$P_1 = A_1 \tag{7.47}$$

$$S_1(t_1) = A_1 + B_1(t_1) + C_1(t_1)^2 + D_1(t_1)^3 + E_1(t_1)^4 \tag{7.48}$$

$$P_2 = A_1 + B_1 t_1 + C_1 t_1^2 + D_1 t_1^3 + E_1 t_1^4 \tag{7.49}$$

Differentiation is performed on the spline polynomial, after which the velocity conditions are imposed.

$$S'_1(t) = B_1 + 2C_1 t + 3D_1 t^2 + 4E_1 t^3 \tag{7.50}$$

$$S'_1(0) = B_1 + 2C_1(0) + 3D_1(0)^2 + 4E_1(0)^3 \tag{7.51}$$

$$0 = B_1 \tag{7.52}$$

$$S'_1(t_1) = B_1 + 2C_1(t_1) + 3D_1(t_1)^2 + 4E_1(t_1)^3 \tag{7.53}$$

$$P'_2 = B_1 + 2C_1 t_1 + 3D_1 t_1^2 + 4E_1 t_1^3 \tag{7.54}$$

Differentiation is performed on the velocity function of Equation (7.45) so that the known acceleration may be used.

$$S''_1(t) = 2C_1 + 6D_1 t + 12E_1 t^2 \tag{7.55}$$

$$S''_1(0) = 2C_1 + 5D_1(0) + 12E_1(0)^2 \tag{7.56}$$

$$0 = 2C_1 \tag{7.57}$$

$$C_1 = 0 \tag{7.58}$$

Equations (7.47), (7.52) and (7.58) explicitly define coefficients A_1, B_1, and C_1, respectively. These values are substituted into Equations (7.49) and (7.54) leaving two equations to be solved for the two remaining unknowns, coefficients D_1 and E_1.

$$P_2 = P_1 + (0)t_1 + (0)t_1^2 + D_1 t_1^3 + E_1 t_1^4 \tag{7.59}$$

$$P'_2 = (0) + 2(0)t_1 + 3D_1 t_1^2 + 4E_1 t_1^3 \tag{7.60}$$

$$P_2 = P_1 + D_1 t_1^3 + E_1 t_1^4 \tag{7.61}$$

$$P'_2 = 3D_1 t_1^2 + 4E_1 t_1^3 \tag{7.62}$$

Coefficient D_1 is obtained first by multiplying Equation (7.61) by 4, multiplying Equation (7.62) by $-t_1$, and then adding the resultants.

$$4P_2 = 4P_1 + 4D_1 t_1^3 + 4E_1 t_1^4 \tag{7.63}$$

$$-P'_2 t_1 = \qquad -3D_1 t_1^3 - 4E_1 t_1^4 \tag{7.64}$$

$$4P_2 - P'_2 t_1 = 4P_1 + D_1 t_1^3 \tag{7.65}$$

$$D_1 = 4\frac{(P_2 - P_1)}{t_1^3} - \frac{P'_2}{t_1^2} \tag{7.66}$$

Coefficient E_1 is then obtained by multiplying Equation (7.61) by 3 and Equation (7.62) by $-t_1$ and then adding the consequences.

$$3P_2 = 3P_1 + 3D_1t_1^3 + 3E_1t_1^4 \tag{7.67}$$

$$-P'_2t_1 = \qquad -3D_1t_1^3 - 4E_1t_1^4 \tag{7.68}$$

$$3P_2 - P'_2t_1 = 3P_1 - E_1t_1^4 \tag{7.69}$$

$$E_1 = 3\frac{(P_1 - P_2)}{t_1^4} + \frac{P'_2}{t_1^3} \tag{7.70}$$

The acceleration at the end of the first segment was not specified as an initial condition. It shall be set equal to the acceleration at the beginning of the second segment to provide another equation dealing with velocities. The coefficients in the formula for acceleration in Equation (7.55) are substituted for by Equations (7.58), (7.66), and (7.70).

$$S''_1(t_1) = 2C_1 + 6D_1t_1 + 12E_1t_1^2 \tag{7.71}$$

$$S''_1(t_1) = 2(0) + 6\left(4\frac{(P_2 - P_1)}{t_1^3} - \frac{P'_2}{t_1^2}\right)t_1 + 12\left(3\frac{(P_1 - P_2)}{t_1^4} + \frac{P'_2}{t_1^3}\right)t_1^2 \tag{7.72}$$

$$S''_1(t_1) = 24\frac{(P_2 - P_1)}{t_1^2} - 6\frac{P'_2}{t_1} + 36\frac{(P_1 - P_2)}{t_1^2} + 12\frac{P'_2}{t_1} \tag{7.73}$$

$$S''_1(t_1) = 12\frac{(P_1 - P_2)}{t_1^2} + 6\frac{P'_2}{t_1} \tag{7.74}$$

The acceleration at the beginning of the second segment is obtained from Equations (7.26) and (7.20).

$$S''_2(t) = 2C_2 + 6D_2 t \tag{7.75}$$

$$S''_2(0) = 2C_2 + 6D_2(0) \tag{7.76}$$

$$S''_2(0) = 2\left(3\frac{(P_3 - P_2)}{t_2^2} - \frac{(P'_3 + 2P'_2)}{t_2}\right) \tag{7.77}$$

The ending acceleration of segment 1 and the beginning acceleration of segment 2 are then set equal.

$$S''_1(t_1) = S''_2(0) \tag{7.78}$$

$$12\frac{(P_1 - P_2)}{t_1^2} + 6\frac{P'_2}{t_1} = 2\left(3\frac{(P_3 - P_2)}{t_2^2} - \frac{(P'_3 + 2P'_2)}{t_2}\right) \tag{7.79}$$

$$6\frac{(P_1 - P_2)}{t_1^2} + 3\frac{P'_2}{t_1} = 3\frac{(P_3 - P_2)}{t_2^2} - \frac{(P'_3 + 2P'_2)}{t_2} \tag{7.80}$$

$$\frac{(P'_3 + 2P'_2)}{t_2} + 3\frac{P'_2}{t_1} = 3\frac{(P_3 - P_2)}{t_2^2} - 6\frac{(P_1 - P_2)}{t_1^2} \tag{7.81}$$

$$(P'_3 + 2P'_2)t_1 + 3P'_2 t_2 = 3(P_3 - P_2)\frac{t_1}{t_2} - 6(P_1 - P_2)\frac{t_2}{t_1} \tag{7.82}$$

$$t_1 P'_3 + (2t_1 + 3t_2)P'_2 = 3(P_3 - P_2)\frac{t_1}{t_2} + 6(P_2 - P_1)\frac{t_2}{t_1} \tag{7.83}$$

d. The Last Spline Segment. The derivations for the last spline segment parallel those of the first segment. It shall likewise have five boundary conditions and thus be a quartic polynomial. As before, the boundary conditions state that the spline must pass through its endpoints.

$$S_{n-1}(0) = P_{n-1} \tag{7.84}$$

$$S_{n-1}(t_{n-1}) = P_n \tag{7.85}$$

The velocity at the starting endpoint of the segment, P_{n-1}, is still undetermined, while the velocity at the spline's end becomes zero as the manipulator must come to a stop.

$$S'_{n-1}(0) = P'_{n-1} \tag{7.86}$$

$$S'_{n-1}(t_{n-1}) = 0 \tag{7.87}$$

The fifth condition states that the acceleration at the end of the segment must become zero, again because the manipulator is stopped.

$$S''_{n-1}(t_{n-1}) = 0 \tag{7.88}$$

Substitutions from the boundary conditions, starting with the endpoint specifications, will obtain the coefficients of $S_{n-1}(t)$.

$$S_{n-1}(t) = A_{n-1} + B_{n-1}t + C_{n-1}t^2 + D_{n-1}t^3 + E_{n-1}t^4 \tag{7.89}$$

$$S_{n-1}(0) = A_{n-1} + B_{n-1}(0) + C_{n-1}(0)^2 + D_{n-1}(0)^3 + E_{n-1}(0)^4 \tag{7.90}$$

$$P_{n-1} = A_{n-1} \tag{7.91}$$

$$S_{n-1}(t_{n-1}) = A_{n-1} + B_{n-1}t_{n-1} + C_{n-1}t_{n-1}^2 + D_{n-1}t_{n-1}^3 + E_{n-1}t_{n-1}^4 \tag{7.92}$$

$$P_n = A_{n-1} + B_{n-1}t_{n-1} + C_{n-1}t_{n-1}^2 + D_{n-1}t_{n-1}^3 + E_{n-1}t_{n-1}^4 \tag{7.93}$$

The spline polynomial is differentiated prior to use of the velocity constraints.

$$S'_{n-1}(t) = B_{n-1} + 2C_{n-1}t + 3D_{n-1}t^2 + 4E_{n-1}t^3 \tag{7.94}$$

$$S'_{n-1}(0) = B_{n-1} + 2C_{n-1}(0) + 3D_{n-1}(0)^2 + 4E_{n-1}(0)^3 \tag{7.95}$$

$$P'_{n-1} = B_{n-1} \tag{7.96}$$

$$S'_{n-1}(t_{n-1}) = B_{n-1} + 2C_{n-1}(t_{n-1}) + 3D_{n-1}(t_{n-1})^2 + 4E_{n-1}(t_{n-1})^3 \tag{7.97}$$

$$0 = B_{n-1} + 2C_{n-1}t_{n-1} + 3D_{n-1}t_{n-1}^2 + 4E_{n-1}t_{n-1}^3 \tag{7.98}$$

Finally, the acceleration condition calls for the spline to be differentiated a second time.

$$S''_{n-1}(t) = 2C_{n-1} + 6D_{n-1}t + 12E_{n-1}t^2 \tag{7.99}$$

$$S''_{n-1}(t_{n-1}) = 2C_{n-1} + 6D_{n-1}(t_{n-1}) + 12E_{n-1}(t_{n-1})^2 \tag{7.100}$$

$$0 = 2C_{n-1} + 6D_{n-1}t_{n-1} + 12E_{n-1}t_{n-1}^2 \tag{7.101}$$

Coefficients A_{n-1} and B_{n-1} are explicitly defined by Equations (7.91) and (7.96), respectively. Their values are substituted into Equations (7.93), (7.98), and (7.101) leaving three equations to be solved this time for the remaining unknowns, C_{n-1}, D_{n-1}, and E_{n-1}.

$$P_n = P_{n-1} + P'_{n-1}t_{n-1} + C_{n-1}t_{n-1}^2 + D_{n-1}t_{n-1}^3 + E_{n-1}t_{n-1}^4 \tag{7.102}$$

$$0 = P'_{n-1} + 2C_{n-1}t_{n-1} + 3D_{n-1}t_{n-1}^2 + 4E_{n-1}t_{n-1}^3 \tag{7.103}$$

$$0 = 2C_{n-1} + 6D_{n-1}t_{n-1} + 12E_{n-1}t_{n-1}^2 \tag{7.104}$$

Coefficient C_{n-1} shall be obtained first. The process begins by eliminating E_{n-1} from two pairings of the three equations. Equation (7.102) is multiplied by 4, Equation (7.103) is multiplied by $-t_{n-1}$, and the results are added.

$$4P_n = 4P_{n-1} + 4P'_{n-1}t_{n-1} + 4C_{n-1}t_{n-1}^2 + 4D_{n-1}t_{n-1}^3 + 4E_{n-1}t_{n-1}^4 \tag{7.105}$$

$$0 = \quad -P'_{n-1}t_{n-1} - 2C_{n-1}t_{n-1}^2 - 3D_{n-1}t_{n-1}^3 - 4E_{n-1}t_{n-1}^4 \tag{7.106}$$

$$4P_n = 4P_{n-1} + 3P'_{n-1}t_{n-1} + 2C_{n-1}t_{n-1}^2 + D_{n-1}t_{n-1}^3 \tag{7.107}$$

Next, Equation (7.103) is multiplied by 3, Equation (7.104) is multiplied by $-t_{n-1}$, and the results are added.

$$0 = 3P'_{n-1} + 6C_{n-1}t_{n-1} + 9D_{n-1}t_{n-1}^2 + 12E_{n-1}t_{n-1}^3 \tag{7.108}$$

$$0 = \quad -2C_{n-1}t_{n-1} - 6D_{n-1}t_{n-1}^2 - 12E_{n-1}t_{n-1}^3 \tag{7.109}$$

$$0 = 3P'_{n-1} + 4C_{n-1}t_{n-1} + 3D_{n-1}t_{n-1}^2 \tag{7.110}$$

Equations (7.107) and (7.110) are in terms of C_{n-1} and D_{n-1} only. Equation (7.107) is multiplied by 3, Equation (7.110) is multiplied by $-t_{n-1}$, and the results are added to obtain C_{n-1}.

$$12P_n = 12P_{n-1} + 9P'_{n-1}t_{n-1} + 6C_{n-1}t_{n-1}^2 + 3D_{n-1}t_{n-1}^3 \tag{7.111}$$

$$0 = \quad -3P'_{n-1}t_{n-1} - 4C_{n-1}t_{n-1}^2 - 3D_{n-1}t_{n-1}^2 \tag{7.112}$$

$$12P_n = 12P_{n-1} + 6P'_{n-1}t_{n-1} + 2C_{n-1}t_{n-1}^2 \tag{7.113}$$

$$C_{n-1}t_{n-1}^2 = 6P_n - 6P_{n-1} - 3P'_{n-1}t_{n-1} \tag{7.114}$$

$$C_{n-1} = 6\,\frac{(P_n - P_{n-1})}{t_{n-1}^2} - 3\,\frac{P'_{n-1}}{t_{n-1}} \tag{7.115}$$

Coefficient D_{n-1} is obtained by multiplying Equation (7.107) by 2, multiplying Equation (7.110) by $-t_{n-1}$, and adding the results.

$$8P_n = 8P_{n-1} + 6P'_{n-1}t_{n-1} + 4C_{n-1}t_{n-1}^2 + 2D_{n-1}t_{n-1}^3 \tag{7.116}$$

$$0 = \qquad -3P'_{n-1}t_{n-1} - 4C_{n-1}t_{n-1}^2 - 3D_{n-1}t_{n-1}^3 \tag{7.117}$$

$$8P_n = 8P_{n-1} + 3P'_{n-1}t_{n-1} - D_{n-1}t_{n-1}^3 \tag{7.118}$$

$$D_{n-1}t_{n-1}^3 = 8P_{n-1} - 8P_n + 3P'_{n-1}t_{n-1} \tag{7.119}$$

$$D_{n-1} = 8\,\frac{(P_{n-1} - P_n)}{t_{n-1}^3} + 3\,\frac{P'_{n-1}}{t_{n-1}^2} \tag{7.120}$$

Finally, coefficient E_{n-1} is obtained by substitutions for C_{n-1} and D_{n-1} into Equation (7.104).

$$0 = 2C_{n-1} + 6D_{n-1}t_{n-1} + 12E_{n-1}t_{n-1}^2 \tag{7.121}$$

$$\begin{aligned} 0 = 2&\left(6\,\frac{(P_n - P_{n-1})}{t_{n-1}^2} - 3\,\frac{P'_{n-1}}{t_{n-1}}\right) \\ &+ 6\left(8\,\frac{(P_{n-1} - P_n)}{t_{n-1}^3} + 3\,\frac{P'_{n-1}}{t_{n-1}^2}\right)t_{n-1} + 12E_{n-1}t_{n-1}^2 \end{aligned} \tag{7.122}$$

$$6E_{n-1}t_{n-1}^2 = -6\,\frac{(P_n - P_{n-1})}{t_{n-1}^2} + 3\,\frac{P'_{n-1}}{t_{n-1}} - 3\left(8\,\frac{(P_{n-1} - P_n)}{t_{n-1}^2} + 3\,\frac{P'_{n-1}}{t_{n-1}}\right) \tag{7.123}$$

$$2E_{n-1}t_{n-1}^2 = -2\frac{(P_n - P_{n-1})}{t_{n-1}^2} + \frac{P'_{n-1}}{t_{n-1}} - 8\frac{(P_{n-1} - P_n)}{t_{n-1}^2} - 3\frac{P'_{n-1}}{t_{n-1}} \tag{7.124}$$

$$2E_{n-1}t_{n-1}^2 = 6\frac{(P_n - P_{n-1})}{t_{n-1}^2} - 2\frac{P'_{n-1}}{t_{n-1}} \tag{7.125}$$

$$E_{n-1} = 3\frac{(P_n - P_{n-1})}{t_{n-1}^4} - \frac{P'_{n-1}}{t_{n-1}^3} \tag{7.126}$$

The accelerations from the left and right of node $n - 1$ are equated as before to provide a final equation involving velocities. The acceleration at the end of the next to last segment is found by substituting coefficients C_{n-2} and D_{n-2} from Equations (7.20) and (7.25), respectively, into the acceleration formula of Equation (7.26).

$$S''_{n-2}(t) = 2C_{n-2} + 6D_{n-2}t \tag{7.127}$$

$$\begin{aligned} S''_{n-1}(t_{n-2}) = {} & 2\left(3\frac{(P_{n-1} - P_{n-2})}{t_{n-2}^2} - \frac{(P'_{n-1} + 2P'_{n-2})}{t_{n-2}}\right) \\ & + 6\left(2\frac{(P_{n-2} - P_{n-1})}{t_{n-2}^3} + \frac{(P'_{n-2} + P'_{n-1})}{t_{n-2}^2}\right)t_{n-2} \end{aligned} \tag{7.128}$$

$$\begin{aligned} S''_{n-2}(t_{n-2}) = {} & 6\frac{(P_{n-1} - P_{n-2})}{t_{n-2}^2} - 2\frac{(P'_{n-1} + 2P'_{n-2})}{t_{n-2}} \\ & + 12\frac{(P_{n-2} - P_{n-1})}{t_{n-2}^2} + 6\frac{(P'_{n-2} + P'_{n-1})}{t_{n-2}} \end{aligned} \tag{7.129}$$

$$S''_{n-2}(t_{n-2}) = 6\frac{(P_{n-2} - P_{n-1})}{t_{n-2}^2} + 2\frac{(P'_{n-2} + 2P'_{n-1})}{t_{n-2}} \tag{7.130}$$

The acceleration at the beginning of the last interval is found by substituting coefficient C_{n-1} of Equation (7.115) into the acceleration formula of Equation (7.99).

$$S''_{n-1}(t) = 2C_{n-1} + 6D_{n-1}t + 12E_{n-1}t^2 \qquad (7.131)$$

$$S''_{n-1}(0) = 2C_{n-1} + 6D_{n-1}(0) + 12E_{n-1}(0)^2 \qquad (7.132)$$

$$S''_{n-1}(0) = 2\left(6\frac{(P_n - P_{n-1})}{t_{n-1}^2} - 3\frac{P'_{n-1}}{t_{n-1}}\right) \qquad (7.133)$$

The ending acceleration of the next to last segment and the beginning acceleration of the last segment are then set equal.

$$S''_{n-2}(t_{n-1}) = S''_{n-1}(0) \qquad (7.134)$$

$$6\frac{(P_{n-2} - P_{n-1})}{t_{n-2}^2} + 2\frac{(P'_{n-2} + 2P'_{n-1})}{t_{n-2}} = 2\left(6\frac{(P_n - P_{n-1})}{t_{n-1}^2} - 3\frac{P'_{n-1}}{t_{n-1}}\right) \qquad (7.135)$$

$$\frac{(P'_{n-2} + 2P'_{n-1})}{t_{n-2}} + 3\frac{P'_{n-1}}{t_{n-1}} = 6\frac{(P_n - P_{n-1})}{t_{n-1}^2} - 3\frac{(P_{n-2} - P_{n-1})}{t_{n-2}^2} \qquad (7.136)$$

$$(P'_{n-2} + 2P'_{n-1})t_{n-1} + 3P'_{n-1}t_{n-2} = 6(P_n - P_{n-1})\frac{t_{n-2}}{t_{n-1}} - 3(P_{n-2} - P_{n-1})\frac{t_{n-1}}{t_{n-2}} \qquad (7.137)$$

$$(2t_{n-1} + 3t_{n-2})P'_{n-1} + t_{n-1}P'_{n-2} = 6(P_n - P_{n-1})\frac{t_{n-2}}{t_{n-1}} + 3(P_{n-1} - P_{n-2})\frac{t_{n-1}}{t_{n-2}} \qquad (7.138)$$

The coefficients for each of the spline polynomials were seen to be dependent on the spline endpoints and the velocities at the endpoints; thus the velocities must be determined prior to evaluation of the spline coefficients. As stated previously, the two ends of the trajectory have velocities of zero; this leaves $n-2$ unknowns. Equation (7.39) provided $n-4$ equations and Equations (7.83) and (7.138) provide the additional two needed to result in a system of $n-2$ equations in $n-2$ unknowns. These equations may be thought of in the following matrix form.

$$\begin{bmatrix} 2t_1+3t_2 & t_1 & 0 & 0 & . \\ t_3 & 2(t_2+t_3) & t_2 & 0 & . \\ 0 & t_4 & 2(t_3+t_4) & t_3 & . \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ . & . & t_{n-2} & 2(t_{n-3}+t_{n-2}) & t_{n-3} \\ . & . & 0 & t_{n-1} & 2t_{n-1}+3t_{n-2} \end{bmatrix} \begin{bmatrix} P'_2 \\ P'_3 \\ P'_4 \\ P'_5 \\ . \\ P'_{n-3} \\ P'_{n-2} \\ P'_{n-1} \end{bmatrix} = \begin{bmatrix} R_2 \\ R_3 \\ R_4 \\ R_5 \\ . \\ R_{n-3} \\ R_{n-2} \\ R_{n-1} \end{bmatrix} \quad (7.139)$$

The elements of the right hand matrix in Equation (7.139) are defined by the following equations.

$$R_2 = 3\frac{t_1}{t_2}(P_3 - P_2) + 6\frac{t_2}{t_1}(P_2 - P_1) \quad (7.140)$$

$$R_{i+1} = 3\frac{t_i}{t_{i+1}}(P_{i+2} - P_{i+1}) + 3\frac{t_{i+1}}{t_i}(P_{i+1} - P_i) \quad \text{for} \quad 2 \le i \le n-3 \quad (7.141)$$

$$R_{n-1} = 6\frac{t_{n-2}}{t_{n-1}}(P_n - P_{n-1}) + 3\frac{t_{n-1}}{t_{n-2}}(P_{n-1} - P_{n-2}) \quad (7.142)$$

2. Numerical Example. This section shall derive the spline polynomials which define an arbitrarily selected trajectory for the Armatron manipulator. Consider the following set of joint coordinates:

$$P_1 = (0^\circ, 0^\circ, 0^\circ, 0^\circ, 0^\circ) \quad (7.143)$$

$$P_2 = (10^\circ, 6^\circ, 20^\circ, -5^\circ, 180^\circ) \quad (7.144)$$

$$P_3 = (25^\circ, 12^\circ, 40^\circ, -7^\circ, 45^\circ) \quad (7.145)$$

$$P_4 = (0^\circ, 0^\circ, 75^\circ, 22^\circ, 45^\circ) \tag{7.146}$$

$$P_5 = (20^\circ, -3^\circ, 80^\circ, 12^\circ, -90^\circ) \tag{7.147}$$

$$P_6 = (-75^\circ, 10^\circ, 60^\circ, 35^\circ, 90^\circ) \tag{7.148}$$

$$P_7 = (-115^\circ, 25^\circ, 50^\circ, 65^\circ, -35^\circ) \tag{7.149}$$

As there are seven nodes, spline polynomials will be generated for six segments. For each of the six segments, one polynomial will be generated for each of the five Armatron joints, bringing the overall number of polynomials to thirty. For the numerical example, only the trajectory of the first joint is considered.

The first step is the determination of arbitrary time units between nodes. Equation (7.1) defined the amount of this time to be the geometric distance between the nodes in five-dimensional space.

$$t_i = \sqrt{\sum_{j=1}^{5} (\theta_{j_{i+1}} - \theta_{j_i})^2} \tag{7.150}$$

$$t_1 = \sqrt{(10^\circ - 0^\circ)^2 + (6^\circ - 0^\circ)^2 + (20^\circ - 0^\circ)^2 + (-5^\circ - 0^\circ)^2 + (180^\circ - 0^\circ)^2} \tag{7.151}$$

$$t_1 = 181.552 \tag{7.152}$$

Similarly,

$$t_2 = 137.441 \tag{7.153}$$

$$t_3 = 53.245 \tag{7.154}$$

$$t_4 = 136.963 \tag{7.155}$$

$$t_5 = 206.211 \tag{7.156}$$

$$t_6 = 135.831 \tag{7.157}$$

As observed during the derivation of equations, the polynomial coefficients are dependent upon not only the node values and the amount of time between the nodes but also the velocities at the nodes. Equations (7.139) through (7.142) are employed here to determine these velocities.

$$\begin{bmatrix} 2t_1 + 3t_2 & t_1 & 0 & 0 & 0 \\ t_3 & 2(t_2 + t_3) & t_2 & 0 & 0 \\ 0 & t_4 & 2(t_3 + t_4) & t_3 & 0 \\ 0 & 0 & t_5 & 2(t_4 + t_5) & t_4 \\ 0 & 0 & 0 & t_6 & 2t_6 + 3t_5 \end{bmatrix} \begin{bmatrix} P'_2 \\ P'_3 \\ P'_4 \\ P'_5 \\ P'_6 \end{bmatrix} = \begin{bmatrix} R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \end{bmatrix} \tag{7.158}$$

$$R_2 = 3\frac{t_1}{t_2}(P_3 - P_2) + 6\frac{t_2}{t_1}(P_2 - P_1) \tag{7.159}$$

$$R_2 = 3\frac{181.552}{137.441}(25 - 10) + 6\frac{137.441}{181.552}(10 - 0) = 104.865 \tag{7.160}$$

$$R_{n-1} = 6\frac{t_{n-2}}{t_{n-1}}(P_n - P_{n-1}) + 3\frac{t_{n-1}}{t_{n-2}}(P_{n-1} - P_{n-2}) \tag{7.161}$$

$$R_6 = 6\frac{206.211}{135.831}(-115 - (-75)) + 3\frac{135.831}{206.211}(-75 - 20) = -552.084 \tag{7.162}$$

$$R_{i+1} = 3\frac{t_i}{t_{i+1}}(P_{i+2} - P_{i+1}) + 3\frac{t_{i+1}}{t_i}(P_{i+1} - P_i) \quad \text{for } 2 \le i \le n-3 \tag{7.163}$$

$$R_3 = 3\frac{t_2}{t_3}(P_4 - P_3) + 3\frac{t_3}{t_2}(P_3 - P_2) \tag{7.164}$$

$$R_3 = 3\,\frac{137.441}{53.245}\,(0-25) + 3\,\frac{53.245}{137.441}\,(25-10) = -176.164 \tag{7.165}$$

$$R_4 = 3\,\frac{t_3}{t_4}\,(P_5 - P_4) + 3\,\frac{t_4}{t_3}\,(P_4 - P_3) \tag{7.166}$$

$$R_4 = 3\,\frac{53.245}{136.963}\,(20-0) + 3\,\frac{136.963}{53.245}\,(0-25) = -169.598 \tag{7.167}$$

$$R_5 = 3\,\frac{t_4}{t_5}\,(P_6 - P_5) + 3\,\frac{t_5}{t_4}\,(P_5 - P_4) \tag{7.168}$$

$$R_5 = 3\,\frac{136.963}{206.211}\,(-75-20) + 3\,\frac{206.211}{136.963}\,(20-0) = -98.958 \tag{7.169}$$

$$\begin{bmatrix} 775.427 & 181.552 & 0 & 0 & 0 \\ 53.245 & 381.372 & 137.441 & 0 & 0 \\ 0 & 136.963 & 380.416 & 53.245 & 0 \\ 0 & 0 & 206.211 & 686.348 & 136.963 \\ 0 & 0 & 0 & 135.831 & 890.295 \end{bmatrix} \begin{bmatrix} P'_2 \\ P'_3 \\ P'_4 \\ P'_5 \\ P'_6 \end{bmatrix} = \begin{bmatrix} 104.865 \\ -176.164 \\ -169.598 \\ -98.958 \\ -552.084 \end{bmatrix} \tag{7.170}$$

$$\begin{bmatrix} 775.427 & 181.552 & 0 & 0 & 0 \\ 0 & 368.906 & 137.441 & 0 & 0 \\ 0 & 0 & 329.389 & 53.245 & 0 \\ 0 & 0 & 0 & 653.014 & 136.963 \\ 0 & 0 & 0 & 0 & 861.806 \end{bmatrix} \begin{bmatrix} P'_2 \\ P'_3 \\ P'_4 \\ P'_5 \\ P'_6 \end{bmatrix} = \begin{bmatrix} 104.865 \\ -183.365 \\ -101.520 \\ -35.402 \\ -544.720 \end{bmatrix} \tag{7.171}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P'_2 \\ P'_3 \\ P'_4 \\ P'_5 \\ P'_6 \end{bmatrix} = \begin{bmatrix} 0.224 \\ -0.377 \\ -0.321 \\ 0.078 \\ -0.632 \end{bmatrix} \tag{7.172}$$

With the node velocities in hand, the coefficients of the polynomials may be determined. Equations (7.8), (7.13), (7.20), and (7.25) state the formulas for the coefficients of the constant, linear, quadratic, and cubic terms, respectively, of the internal spline polynomials.

$$A_i = P_i \tag{7.173}$$

$$B_i = P'_i \tag{7.174}$$

$$C_i = 3\frac{P_{i+1}}{t_i^2} - \frac{P'_{i+1} + 2P'_i}{t_i} \tag{7.175}$$

$$D_i = 2\frac{P_i - P_{i+1}}{t_i^3} + \frac{P'_i + P'_{i+1}}{t_i^2} \tag{7.176}$$

$$A_2 = 10 \tag{7.177}$$

$$B_2 = 0.224 \tag{7.178}$$

$$C_2 = 3\frac{25 - 10}{(137.441)^2} - \frac{-0.377 + 2(0.224)}{137.441} = 1.866\text{E-}3 \tag{7.179}$$

$$D_2 = 2\frac{10 - 25}{(137.441)^3} + \frac{0.224 + (-0.377)}{(137.441)^2} = -1.965\text{E-}5 \tag{7.180}$$

$$P_2(t) = 10 + (0.224)t + (1.866\text{E-}3)t^2 + (-1.965\text{E-}5)t^3 \tag{7.181}$$

Substitution of t_2 into Equation (7.181) provides a quick check to show that the polynomial does indeed pass through the segment end value of 25.

$$P_2(137.441) = 25.019 \tag{7.182}$$

The remaining intermediate polynomials are obtained and checked in similar fashion.

$$A_3 = 25 \tag{7.183}$$

$$B_3 = -0.377 \tag{7.184}$$

$$C_3 = 3\frac{0-25}{(53.245)^2} - \frac{-0.321 + 2(-0.377)}{53.245} = -6.265\text{E-}3 \tag{7.185}$$

$$D_3 = 2\frac{25-0}{(53.245)^3} + \frac{-0.377 + (-0.321)}{(53.245)^2} = 8.503\text{E-}5 \tag{7.186}$$

$$P_3(t) = 25 + (-0.377)t + (-6.265\text{E-}3)t^2 + (8.503\text{E-}5)t^3 \tag{7.187}$$

$$P_3(53.245) = 0.001 \tag{7.188}$$

$$A_4 = 0 \tag{7.189}$$

$$B_4 = -0.321 \tag{7.190}$$

$$C_4 = 3\frac{20-0}{(136.963)^2} - \frac{0.078 + 2(-0.321)}{136.963} = 7.316\text{E-}3 \tag{7.191}$$

$$D_4 = 2\frac{0-20}{(136.963)^3} + \frac{-0.321 + 0.078}{(136.963)^2} = -2.852\text{E-}5 \tag{7.192}$$

$$P_4(t) = 0 + (-0.321)t + (7.316\text{E-}3)t^2 + (-2.852\text{E-}5)t^3 \tag{7.193}$$

$$P_4(136.963) = 19.999 \tag{7.194}$$

$$A_5 = 20 \tag{7.195}$$

$$B_5 = 0.078 \tag{7.196}$$

$$C_5 = 3\frac{-75 - 20}{(206.211)^2} - \frac{-0.632 + 2(0.078)}{206.211} = -4.394\text{E-}3 \tag{7.197}$$

$$D_5 = 2\frac{20 - (-75)}{(206.211)^3} + \frac{0.078 + (-0.632)}{(206.211)^2} = 8.640\text{E-}6 \tag{7.198}$$

$$P_5(t) = 20 + (0.078)t + (-4.394\text{E-}3)t^2 + (8.640\text{E-}6)t^3 \tag{7.199}$$

$$P_5(206.211) = -75.000 \tag{7.200}$$

Each internal polynomial is observed to produce a value near its desired endpoint at the end of its time interval.

Equations (7.47), (7.52), (7.58), (7.66), and (7.70) define the coefficients of the first spline polynomial's constant, linear, quadratic, cubic, and quartic terms, respectively.

$$A_1 = P_1 \tag{7.201}$$

$$A_1 = 0 \tag{7.202}$$

$$B_1 = 0 \tag{7.203}$$

$$C_1 = 0 \tag{7.204}$$

$$D_1 = 4\frac{P_2 - P_1}{t_1^3} - \frac{P'_2}{t_1^2} \tag{7.205}$$

$$D_1 = 4\frac{10 - 0}{(181.552)^3} - \frac{0.224}{(181.552)^2} = -1.116\text{E-}7 \tag{7.206}$$

$$E_1 = 3\frac{P_1 - P_2}{t_1^4} + \frac{P'_2}{t_1^3} \tag{7.207}$$

$$E_1 = 3\frac{0 - 10}{(181.552)^4} + \frac{0.224}{(181.552)^3} = 9.819\text{E-}9 \tag{7.208}$$

$$P_1(t) = 0 + (0)t + (0)t^2 + (-1.116\text{E-}7)t^3 + (9.819\text{E-}9)t^4 \tag{7.209}$$

$$P_1(181.552) = 10.000 \tag{7.210}$$

Equations (7.91), (7.96), (7.115), (7.120), and (7.126) define the coefficients of the terms in increasing order of the last spline polynomial.

$$A_{n-1} = P_{n-1} \tag{7.211}$$

$$A_6 = -75 \tag{7.212}$$

$$B_{n-1} = P'_{n-1} \tag{7.213}$$

$$B_6 = -0.632 \tag{7.214}$$

$$C_{n-1} = 6\frac{(P_n - P_{n-1})}{t_{n-1}^2} - 3\frac{P'_{n-1}}{t_{n-1}} \tag{7.215}$$

$$C_6 = 6\frac{(-115 - (-75))}{(135.831)^2} - 3\frac{-0.632}{135.831} = 9.504\text{E-}4 \tag{7.216}$$

$$D_{n-1} = 8\,\frac{P_{n-1} - P_n}{t_{n-1}^3} + 3\,\frac{P'_{n-1}}{t_{n-1}^2} \tag{7.217}$$

$$D_6 = 8\,\frac{-75 - (-115)}{(135.831)^3} + 3\,\frac{-0.632}{(135.831)^2} = 2.493\text{E-5} \tag{7.218}$$

$$E_{n-1} = 3\,\frac{P_n - P_{n-1}}{t_{n-1}^4} - \frac{P'_{n-1}}{t_{n-1}^3} \tag{7.219}$$

$$E_6 = 3\,\frac{-115 - (-75)}{(135.831)^4} - \frac{-0.632}{(135.831)^3} = -1.003\text{E-7} \tag{7.220}$$

$$P_6(t) = -75 + (-0.632)t + (9.504\text{E-4})t^2 + (2.493\text{E-5})t^3 + (-1.003\text{E-7})t^4 \tag{7.221}$$

$$P_6(135.831) = -114.976 \tag{7.222}$$

The spline polynomials for joint variables θ_2, θ_3, θ_4, and θ_5 are obtained in a parallel fashion.

Figure 7.2 graphs the information obtained about the trajectory in this section. The nodal velocities are used to give some indication as to the slope of the spline as it passes through the nodes. The values used are those obtained by the program example.

3. Program Control. The input of nodes along the desired trajectory is controlled by procedure `nodes_and_distances`. This procedure determines the distances, or scale times, between nodes also. The body of the procedure follows.

```
dsply_nodes_dists (&row, cols);
n = input_nodes (theta, row, cols);
calc_distance (n, theta, t, row, cols[6]);
wait_then_erase (10);
return (n);
```

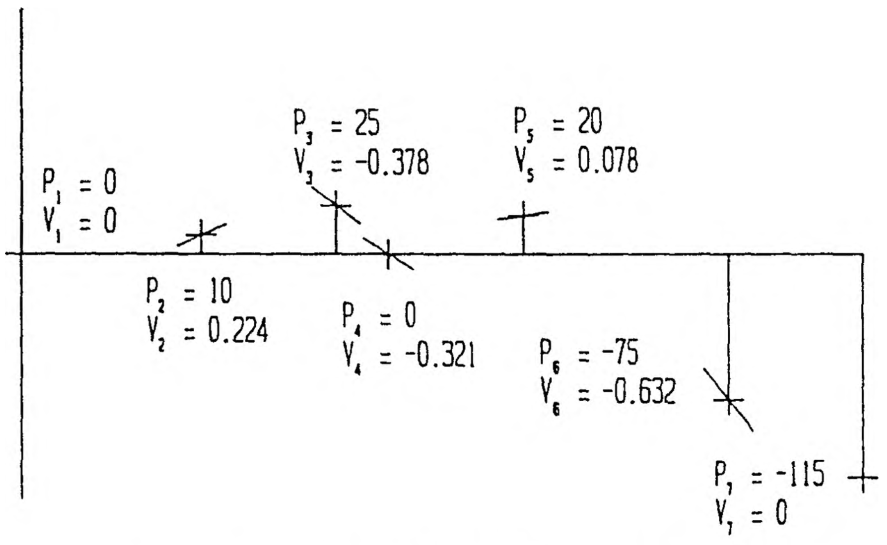


Figure 7.2. Trajectory Nodal Positions and Velocities

The determination of the spline polynomials is performed by procedure `calc_polynomials`. This process is performed in two steps by procedures `calc_node_velocities` and `calc_coefficients`; these two procedures form the entire body of `calc_polynomials`, as follows.

```
calc_node_velocities (n, p, t, vel);
calc_coefficients (n, p, t, vel, a, b, c, d, e);
```

Procedure `calc_node_velocities` in turn invokes procedures to follow the steps described in the derivations and examples of this section. The procedures themselves are straightforward.

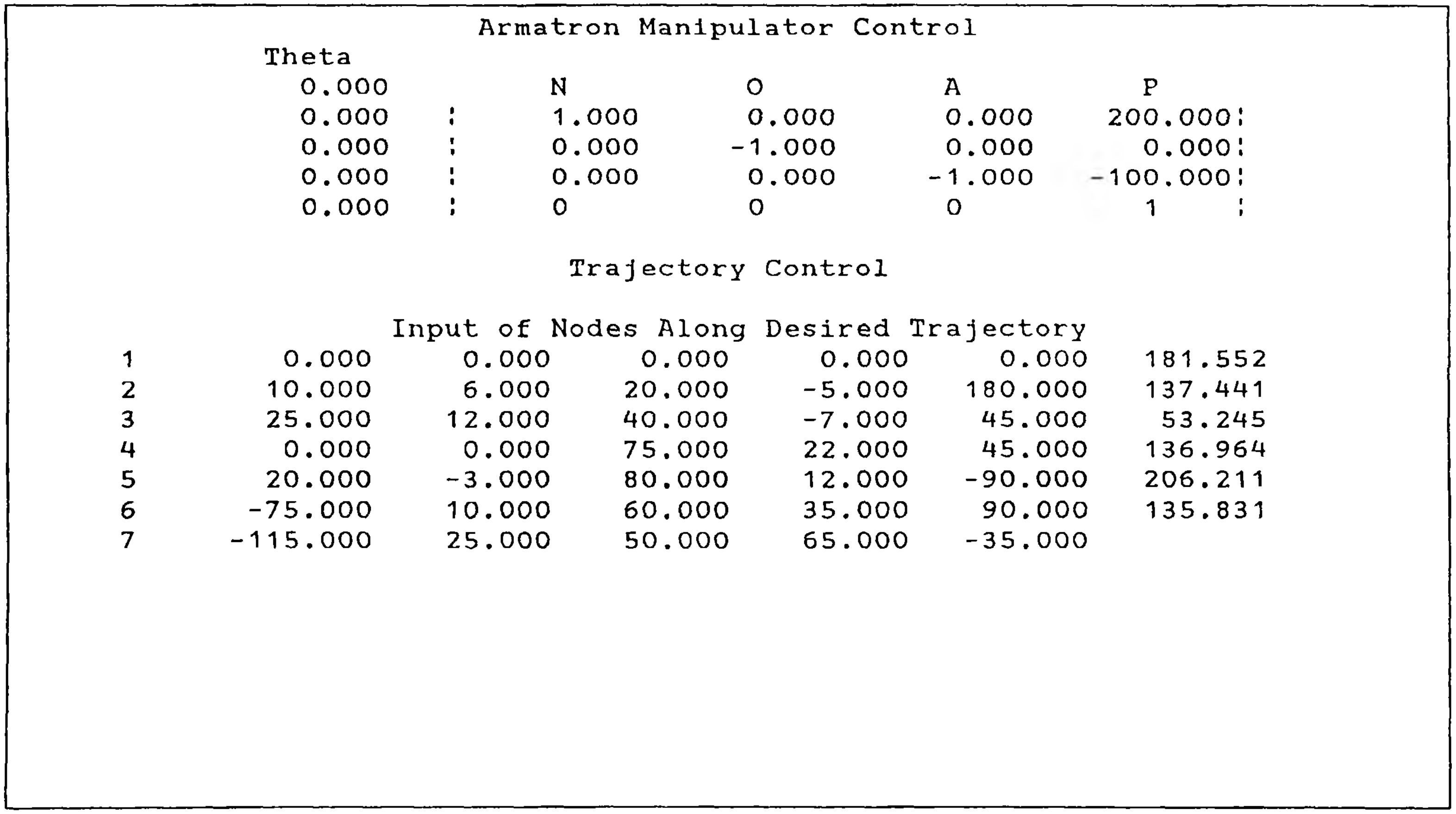
```
dsply_node_velocities (&row, cols);
equate_quartic_cubic_accs (t, p, coeff, rhs, row, cols);
equate_cubic_accs (n, t, p, coeff, rhs, row, cols);
equate_cubic_quartic_accs (n, t, p, coeff, rhs, row, cols);
wait_then_continue ( );
forward_eliminate_term1 (n, coeff, rhs, row, cols);
wait_then_continue ( );
backward_eliminate_term3 (n, coeff, rhs, vel, row, cols);
wait_then_erase (10);
```

Procedure `calc_coefficients` determines the polynomial factors in three groups: starting, intermediate, and ending.

```
dsply_coefficients (&row, cols);
calc_starting_quartic       (p, t, vel, a, b, c, d, e, row, cols);
calc_intermediate_cubics (n, p, t, vel, a, b, c, d, e, row, cols);
calc_ending_quartic      (n, p, t, vel, a, b, c, d, e, row, cols);
```

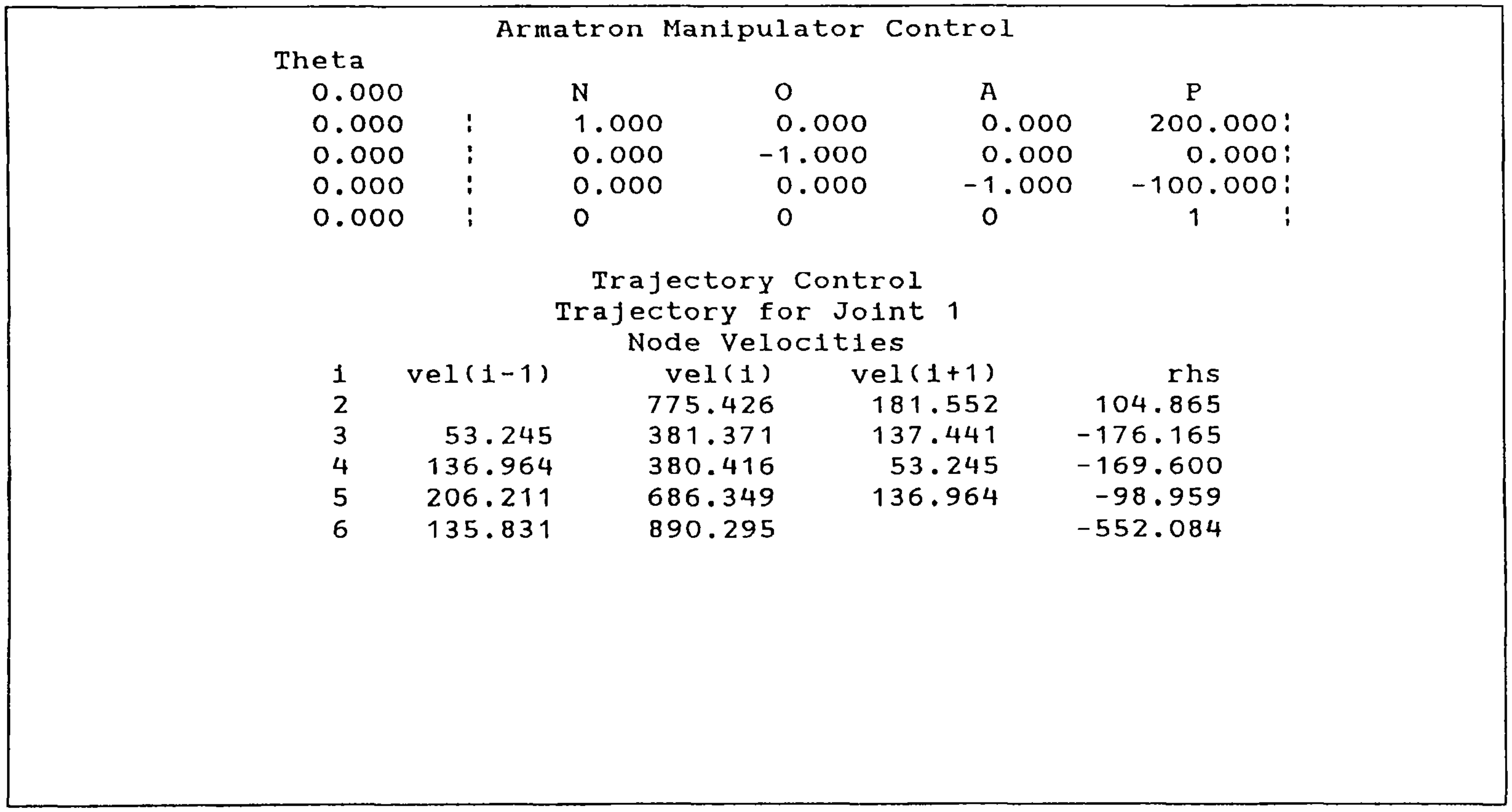
```
    wait_then_erase (10);
```

4. Program Example. Figure 7.4 shows the display observed during the input of joint variable sets. There is virtually no difference here between the distances calculated in Equations (7.152) through (7.157) and those shown in the figure. Figure 7.4 gives the display reflecting the calculated matrix equation for the determination of the nodal velocities; comparison with Equation (7.170) shows only insignificant differences in values. Figures 7.5 and 7.6 reflect the status of the matrix equation after the forward and backward elimination steps, respectively. Comparison with Equations (7.171) and (7.172) again shows only insignificant differences. Finally, Figure (7.7) shows the display of polynomial coefficients for the spline polynomials of the first joint variable. Comparison with the coefficients of Equations (7.209) , (7.181), (7.187), (7.193), (7.199), and (7.221) will show that an increasing amount of precision has been lost in the numerical example.



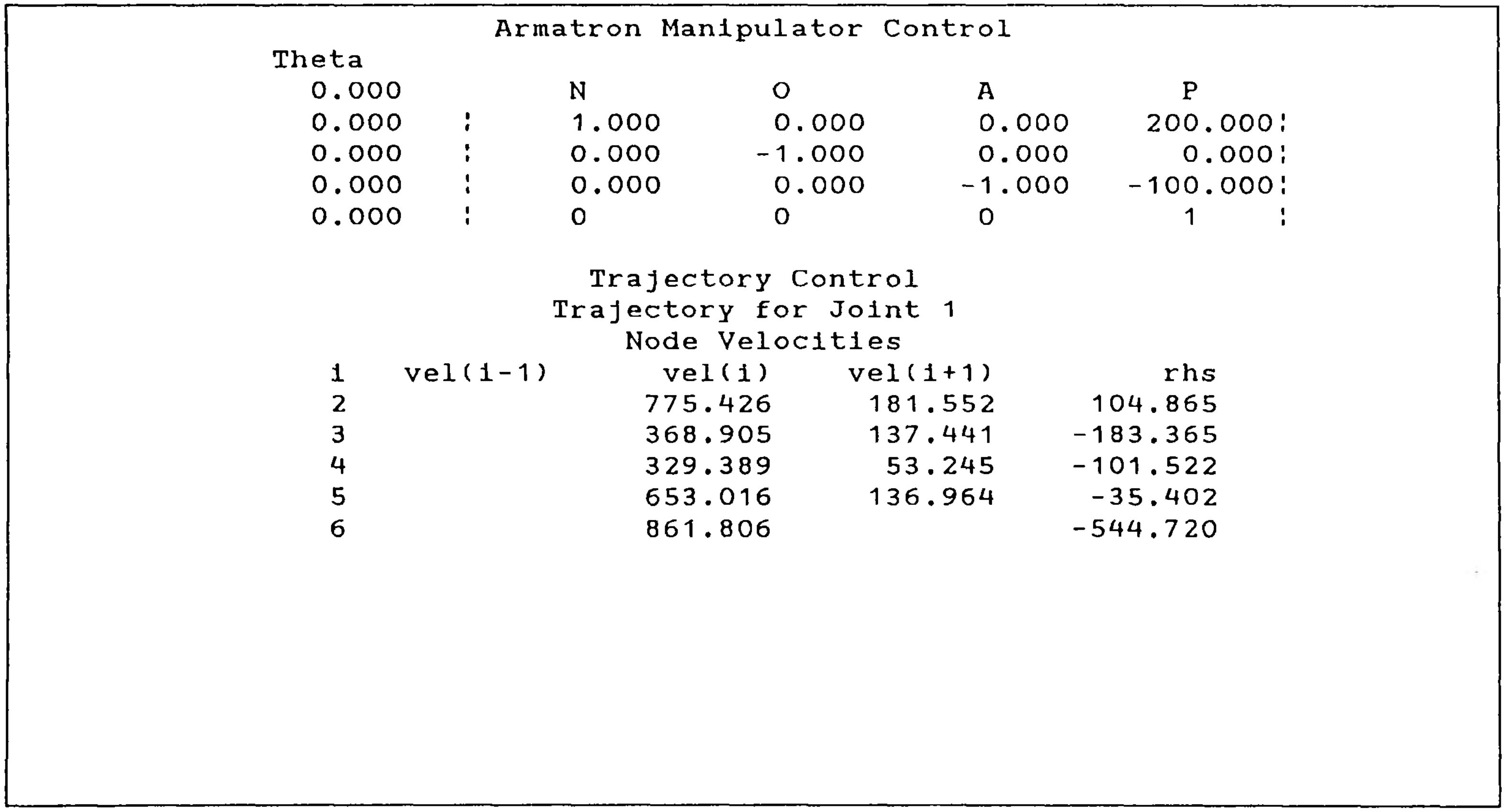
```
                       Armatron Manipulator Control
             Theta
               0.000              N          O          A          P
               0.000   :        1.000      0.000      0.000    200.000:
               0.000   :        0.000     -1.000      0.000      0.000:
               0.000   :        0.000      0.000     -1.000   -100.000:
               0.000   :        0              0          0          1     :

                                Trajectory Control

                    Input of Nodes Along Desired Trajectory
      1          0.000      0.000      0.000      0.000      0.000    181.552
      2         10.000      6.000     20.000     -5.000    180.000    137.441
      3         25.000     12.000     40.000     -7.000     45.000     53.245
      4          0.000      0.000     75.000     22.000     45.000    136.964
      5         20.000     -3.000     80.000     12.000    -90.000    206.211
      6        -75.000     10.000     60.000     35.000     90.000    135.831
      7       -115.000     25.000     50.000     65.000    -35.000
```

Figure 7.3. Input of Nodes and Scale Times



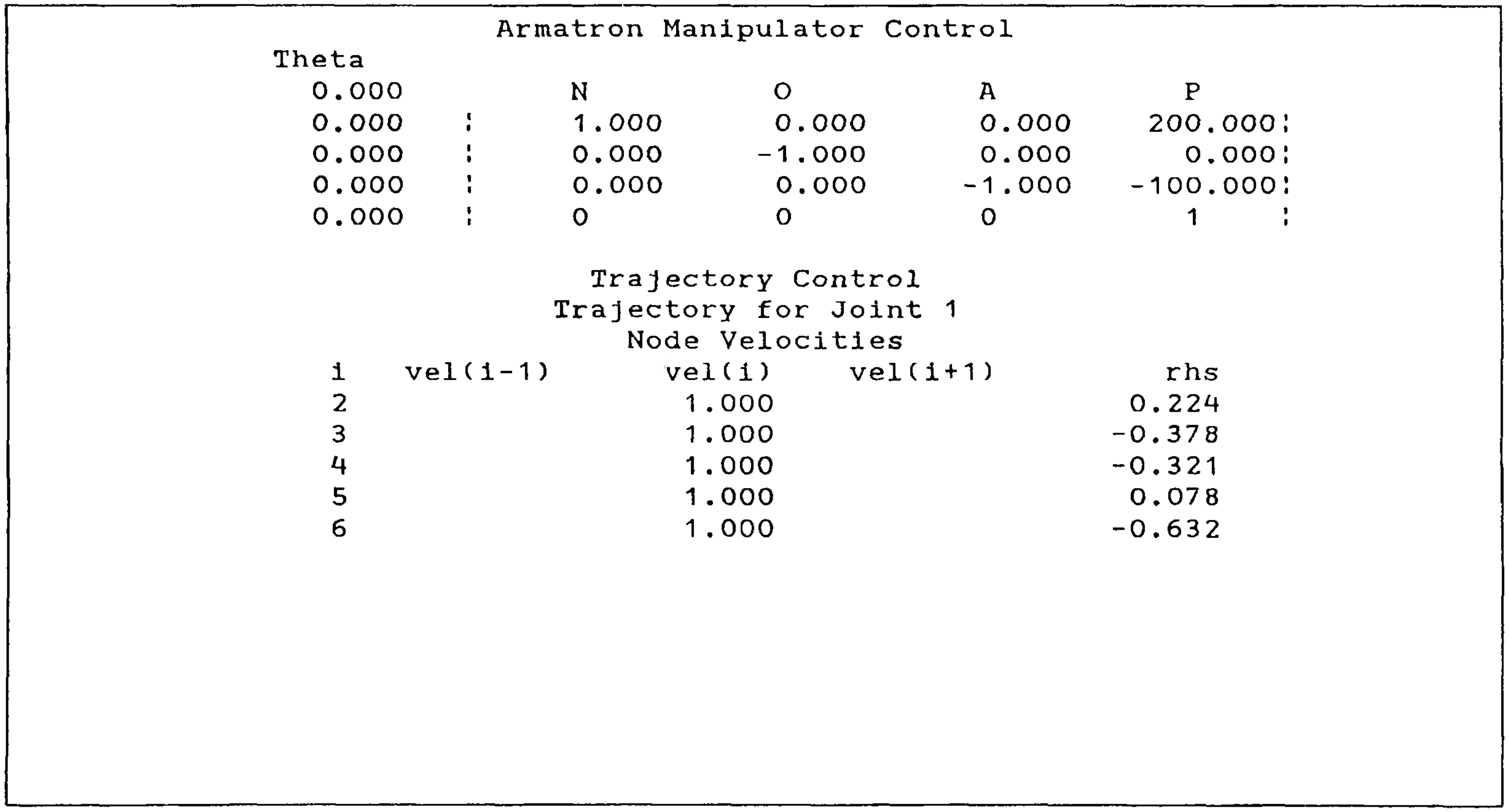
```
                    Armatron Manipulator Control
      Theta
        0.000           N          O          A          P
        0.000    :    1.000      0.000      0.000    200.000:
        0.000    :    0.000     -1.000      0.000      0.000:
        0.000    :    0.000      0.000     -1.000   -100.000:
        0.000    :    0 	         0          0          1    :

                         Trajectory Control
                       Trajectory for Joint 1
                           Node Velocities
         i   vel(i-1)      vel(i)     vel(i+1)          rhs
         2                775.426     181.552       104.865
         3     53.245     381.371     137.441      -176.165
         4    136.964     380.416      53.245      -169.600
         5    206.211     686.349     136.964       -98.959
         6    135.831     890.295                  -552.084
```

Figure 7.4. Node Velocity Equation Coefficients



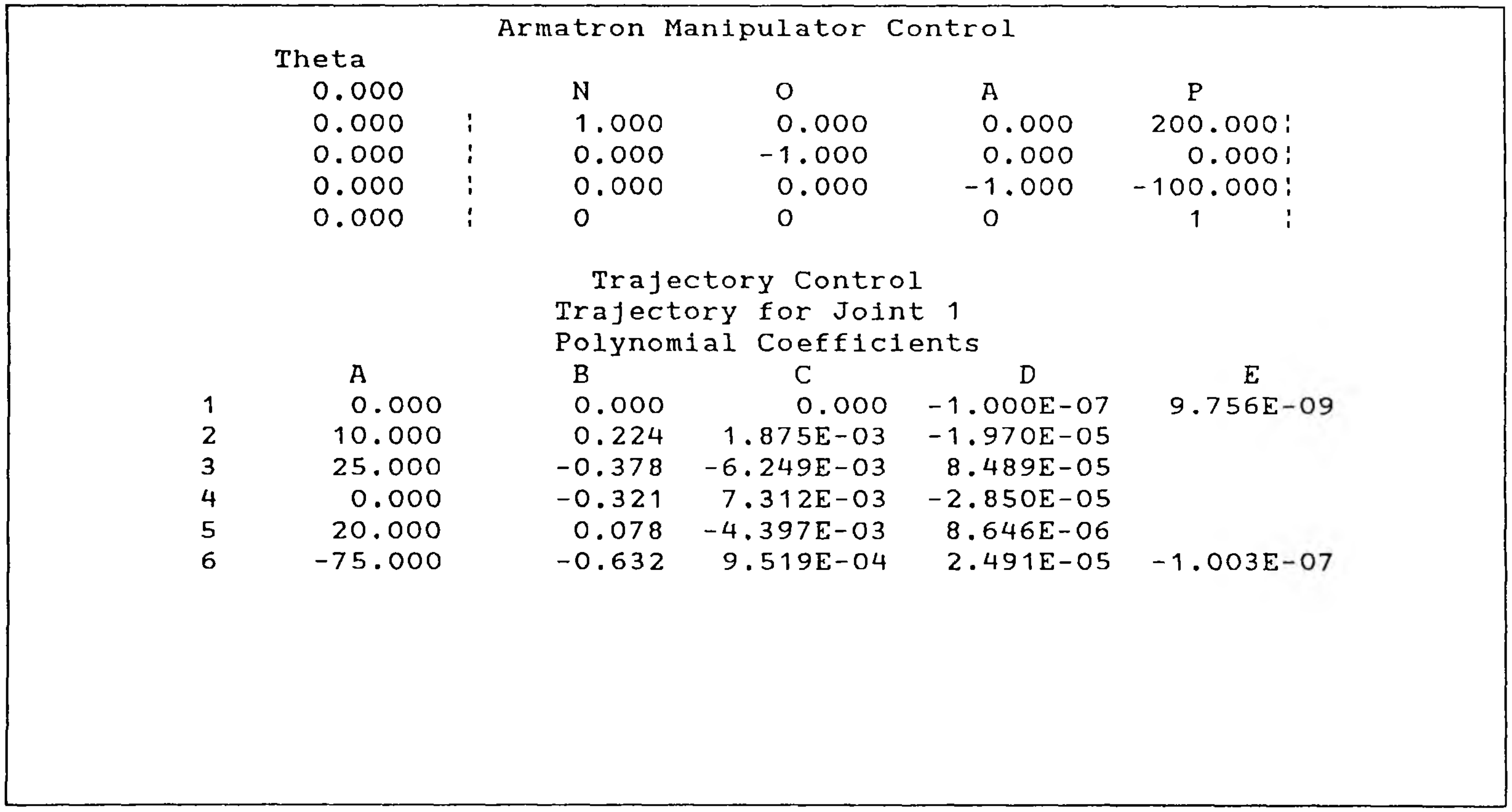
```
                    Armatron Manipulator Control
             Theta
              0.000             N          O          A          P
              0.000   :       1.000      0.000      0.000    200.000:
              0.000   :       0.000     -1.000      0.000      0.000:
              0.000   :       0.000      0.000     -1.000   -100.000:
              0.000   :       0                     0          1    :

                            Trajectory Control
                          Trajectory for Joint 1
                              Node Velocities
               i    vel(i-1)       vel(i)     vel(i+1)          rhs
               2                 775.426      181.552       104.865
               3                 368.905      137.441      -183.365
               4                 329.389       53.245      -101.522
               5                 653.016      136.964       -35.402
               6                 861.806                   -544.720
```

Figure 7.5. Node Velocity Equations After Forward Elimination



```
                 Armatron Manipulator Control
      Theta
        0.000             N          O          A          P
        0.000    :      1.000      0.000      0.000    200.000:
        0.000    :      0.000     -1.000      0.000      0.000:
        0.000    :      0.000      0.000     -1.000   -100.000:
        0.000    :      0   0          0          1      :

                      Trajectory Control
                    Trajectory for Joint 1
                        Node Velocities
         i    vel(i-1)       vel(i)     vel(i+1)          rhs
         2                   1.000                      0.224
         3                   1.000                     -0.378
         4                   1.000                     -0.321
         5                   1.000                      0.078
         6                   1.000                     -0.632
```

Figure 7.6. Node Velocity Equations After Backward Elimination



```
                    Armatron Manipulator Control
     Theta
       0.000             N          O          A          P
       0.000   |       1.000      0.000      0.000    200.000|
       0.000   |       0.000     -1.000      0.000      0.000|
       0.000   |       0.000      0.000     -1.000   -100.000|
       0.000   |       0 0        0          0          1    |

                         Trajectory Control
                       Trajectory for Joint 1
                       Polynomial Coefficients
            A            B           C           D           E
  1       0.000        0.000       0.000  -1.000E-07   9.756E-09
  2      10.000        0.224   1.875E-03  -1.970E-05
  3      25.000       -0.378  -6.249E-03   8.489E-05
  4       0.000       -0.321   7.312E-03  -2.850E-05
  5      20.000        0.078  -4.397E-03   8.646E-06
  6     -75.000       -0.632   9.519E-04   2.491E-05  -1.003E-07
```

Figure 7.7. Spline Polynomial Function Coefficients

B. EXAMINATION OF SPLINE EXTREMA

With the spline polynomials now determined, it must be guaranteed that the trajectories specified by them will not require the joint variables to attain values outside of their physical constraints. This test is performed by examining the velocity functions of the polynomials. As long as the velocity along the trajectory is increasing or decreasing, the joint positioning shall continue to do likewise. When the velocity becomes zero, however, it is possible that the positioning has reached an extremum. For example, if the velocity had been increasing and eventually became zero, the velocity could become negative and the positioning would start to decrease; in this case, a maximum value was attained. On the other hand, the velocity might increase, become zero, and then begin to increase again; in this case, no maximum is present. Only the maxima and minima attained by the polynomial need be examined, and these occur only when the velocity function of the spline polynomial becomes zero.

1. Derivation of Equations. The velocity along a defined trajectory is obtained at any point by an evaluation of the first derivative of the spline polynomial. The times at which the velocity becomes zero are then obtained by solving the equation of the velocity function and zero. The obtained times must then be compared to the time for which the manipulator is traversing the trajectory segment in question; if the time is outside of this window, no further examination need be made. As the trajectory was constructed to be continuous through the second derivative at the nodes, no examinations need be made at the nodes themselves either. When the critical times have been determined, the spline polynomials are evaluated at them to determine if any out of range conditions will result from an attempt to follow the trajectory. If this is the case, corrective measures must be taken, such as adding additional path nodes to prevent the exceedance.

a. Internal Spline Segments. The velocity function for the i^{th} internal spline segment was given by Equation (7.11). The function is set equal to zero to obtain the desired times.

$$S'_i(t) = B_i + 2C_i + 3D_it^2 \tag{7.223}$$

$$(3D_i)t_{cp_i}^2 + (2C_i)t_{cp_i} + (B_i) = 0 \tag{7.224}$$

$$t_{cp_i} = \frac{-(2C_i) \pm \sqrt{(2C_i)^2 - 4(3D_i)(B_i)}}{2(3D_i)} \tag{7.225}$$

$$t_{cp_i} = \frac{-2C_i \pm \sqrt{4C_i^2 - 4(3B_iD_i)}}{2(3D_i)} \tag{7.226}$$

$$t_{cp_i} = \frac{-C_i \pm \sqrt{C_i^2 - 3B_iD_i}}{3D_i} \tag{7.227}$$

Let the discriminant in the formula for the critical time be represented by d_i.

$$d_i = C_i^2 - 3B_iD_i \tag{7.228}$$

If discriminant d_i is negative, there are no solutions to Equation (7.224) and consequently no extrema. If the discriminant is positive, the equation has two roots which must be examined. Finally, if the discriminant is zero, Equation (7.224) has a double root and there is only one time at which the polynomial must be investigated.

b. The Terminal Spline Segments. As the first and last spline segments are both fourth order, the solutions for the times of possible extrema occurrences are similar for both. Equation (7.11) defined the velocity function for the first spline segment. The solution of the equation of this function with zero is complicated by the presence of a cubic term. For a cubic equation of the following form

$$t^3 + a_1 t^2 + a_2 t + a_3 = 0 \tag{7.229}$$

the solution is obtained using the following formulae:

$$Q = \frac{3a_2 - a_1^2}{9} \tag{7.230}$$

$$R = \frac{9a_1 a_2 - 27a_3 - 2a_1^3}{54} \tag{7.231}$$

$$S = \sqrt[3]{R + \sqrt{Q^3 + R^2}} \tag{7.232}$$

$$T = \sqrt[3]{R - \sqrt{Q^3 + R^2}} \tag{7.233}$$

$$t_1 = S + T - \frac{a_1}{3} \tag{7.234}$$

$$t_2 = -\frac{1}{2}(S + T) - \frac{a_1}{3} + \frac{1}{2} i\sqrt{3}(S - T) \tag{7.235}$$

$$t_3 = -\frac{1}{2}(S + T) - \frac{a_1}{3} - \frac{1}{2} i\sqrt{3}(S - T) \tag{7.236}$$

The velocity function of Equation (7.11) is set equal to zero and adjusted to the form of Equation (7.229) so that quantities Q and R may be determined.

$$S'_1(t) = B_1 + 2C_1 t + 3D_1 t^2 + 4E_1 t^3 \tag{7.237}$$

$$(4E_1)t_{cp_1}^3 + (3D_1)t_{cp_1}^2 + (2C_1)t_{cp_1} + (B_1) = 0 \tag{7.238}$$

$$t_{cp_1}^3 + \left(\frac{3D_1}{4E_1}\right)t_{cp_1}^2 + \left(\frac{C_1}{2E_1}\right)t_{cp_1} + \left(\frac{B_1}{4E_1}\right) = 0 \tag{7.239}$$

$$Q_1 = \frac{3\left(\frac{C_1}{2E_1}\right) - \left(\frac{3D_1}{4E_1}\right)^2}{9} \qquad (7.240)$$

$$Q_1 = \frac{C_1}{6E_1} - \frac{D_1^2}{16E_1^2} \qquad (7.241)$$

$$Q_1 = \frac{1}{2E_1}\left(\frac{C_1}{3} - \frac{D_1^2}{8E_1}\right) \qquad (7.242)$$

$$R_1 = \frac{9\left(\frac{3D_1}{4E_1}\right)\left(\frac{C_1}{2E_1}\right) - 27\left(\frac{B_1}{4E_1}\right) - 2\left(\frac{3D_1}{4E_1}\right)^3}{54} \qquad (7.243)$$

$$R_1 = \frac{1}{2}\left(\frac{D_1}{4E_1}\right)\left(\frac{C_1}{2E_1}\right) - \frac{1}{2}\left(\frac{B_1}{4E_1}\right) - \frac{D_1^3}{64E_1^3} \qquad (7.244)$$

$$R_1 = \frac{1}{8E_1}\left(\frac{C_1D_1}{2E_1} - B_1 - \frac{D_1^3}{8E_1^2}\right) \qquad (7.245)$$

$$R_1 = \frac{1}{8E_1}\left(\frac{D_1}{2E_1}\left(C_1 - \frac{D_1^2}{4E_1}\right) - B_1\right) \qquad (7.246)$$

Equations (7.232) and (7.233) both require the determination of the square root of $Q^3 + R^2$, so this expression is the discriminant for the solution of the cubic equation.

$$d = Q^3 + R^2 \qquad (7.247)$$

Note that Q must be less than zero in order for d to take on a negative value. If d is positive, both S and T acquire real number values, and their use in Equation (7.234) produces a real number result for t_1. However, their use in Equations (7.235) and

(7.236) produces two complex number solutions; thus there is only one solution to be investigated when d is greater than zero.

$$t_{cp_1} = \sqrt[3]{R_1 + \sqrt{d_1}} + \sqrt[3]{R_1 - \sqrt{d_1}} - \frac{D_1}{4E_1} \tag{7.248}$$

If the discriminant should take on the value 0, then S and T take on the same value, $\sqrt[3]{R}$. Equations (7.235) and (7.236) become the same double root; this double root and the root of Equation (7.234) are the two critical times for this case.

$$t_{cp_{1,1}} = 2\sqrt[3]{R_1} - \frac{D_1}{4E_1} \tag{7.249}$$

$$t_{cp_{1,2}} = -\sqrt[3]{R_1} - \frac{D_1}{4E_1} \tag{7.250}$$

In the third case, the discriminant is less than zero, and both S and T become complex numbers.

$$S = \sqrt[3]{R + \sqrt{-d}\, i} \tag{7.251}$$

$$T = \sqrt[3]{R - \sqrt{-d}\, i} \tag{7.252}$$

Consider the alternative expression for the complex number $a + bi$.

$$a + bi = r(\cos\theta + i\sin\theta) \tag{7.253}$$

$$\theta = \tan^{-1}\left(\frac{b}{a}\right) \tag{7.254}$$

$$r = \sqrt{a^2 + b^2} \tag{7.255}$$

Equations (7.251), (7.252), and (7.254) then become

$$S = \sqrt[3]{\sqrt{R^2 - d}\,(\cos\theta + i\sin\theta)} \tag{7.256}$$

$$T = \sqrt[3]{\sqrt{R^2 - d}\,(\cos\theta - i\sin\theta)} \tag{7.257}$$

$$\theta = \tan^{-1}\left(\frac{\sqrt{-d}}{R}\right) \tag{7.258}$$

An expression of the form $(\cos\theta + i\sin\theta)^n$ may be replaced by deMoivre's formula which states

$$(\cos\theta + i\sin\theta)^n = \cos n\theta + i\sin n\theta \tag{7.259}$$

Application of deMoivre's formula to Equation (7.256), along with a replacement for d from Equation (7.247), simplifies the expression for S.

$$S = \sqrt[3]{\sqrt{R^2 - Q^3 - R^2}}\left(\cos\frac{\theta}{3} + i\sin\frac{\theta}{3}\right) \tag{7.260}$$

$$S = \sqrt{-Q}\left(\cos\frac{\theta}{3} + i\sin\frac{\theta}{3}\right) \tag{7.261}$$

Next, Equation (7.259) is adjusted for use with T.

$$(\cos(-\theta) + i\sin(-\theta))^n = \cos(n(-\theta)) + i\sin(n(-\theta)) \tag{7.262}$$

$$(\cos(-\theta) + i\sin(-\theta))^n = \cos(n\theta) - i\sin(n\theta) \tag{7.263}$$

$$T = \sqrt{-Q}\left(\cos\frac{\theta}{3} - i\sin\frac{\theta}{3}\right) \tag{7.264}$$

The expressions obtained here for S and T are substituted into Equations (7.234), (7.235), and (7.236) to simplify the relationships for the three real roots of the original cubic equation.

$$t_{cp_{1,1}} = \left(\sqrt{-Q_1}\left(\cos\frac{\theta}{3} + i\sin\frac{\theta}{3}\right)\right) + \left(\sqrt{-Q_1}\left(\cos\frac{\theta}{3} - i\sin\frac{\theta}{3}\right)\right) - \frac{3D_1}{3(4E_1)} \quad (7.265)$$

$$t_{cp_{1,1}} = 2\sqrt{-Q_1}\cos\frac{\theta}{3} - \frac{D_1}{4E_1} \quad (7.266)$$

$$t_{cp_{1,2}} = -\frac{1}{2}\left(\sqrt{-Q_1}\left(\cos\frac{\theta}{3} + i\sin\frac{\theta}{3}\right) + \sqrt{-Q_1}\left(\cos\frac{\theta}{3} - i\sin\frac{\theta}{3}\right)\right) - \frac{3D_1}{3(4E_1)} + \frac{1}{2}i\sqrt{3}\left(\sqrt{-Q_1}\left(\cos\frac{\theta}{3} + i\sin\frac{\theta}{3}\right) - \sqrt{-Q_1}\left(\cos\frac{\theta}{3} - i\sin\frac{\theta}{3}\right)\right) \quad (7.267)$$

$$t_{cp_{1,2}} = -\frac{1}{2}\left(2\sqrt{-Q_1}\cos\frac{\theta}{3}\right) - \frac{D_1}{4E_1} + \frac{1}{2}i\sqrt{3}\left(2i\sqrt{-Q_1}\sin\frac{\theta}{3}\right) \quad (7.268)$$

$$t_{cp_{1,2}} = -\sqrt{-Q_1}\cos\frac{\theta}{3} - \frac{D_1}{4E_1} - \sqrt{3}\sqrt{-Q_1}\sin\frac{\theta}{3} \quad (7.269)$$

$$t_{cp_{1,2}} = -\sqrt{-Q_1}\left(\cos\frac{\theta}{3} + \sqrt{3}\sin\frac{\theta}{3}\right) - \frac{D_1}{4E_1} \quad (7.270)$$

$$t_{cp_{1,3}} = -\frac{1}{2}\left(\sqrt{-Q_1}\left(\cos\frac{\theta}{3} + i\sin\frac{\theta}{3}\right) + \sqrt{-Q_1}\left(\cos\frac{\theta}{3} - i\sin\frac{\theta}{3}\right)\right) - \frac{3D_1}{3(4E_1)} - \frac{1}{2}i\sqrt{3}\left(\sqrt{-Q_1}\left(\cos\frac{\theta}{3} + i\sin\frac{\theta}{3}\right) - \sqrt{-Q_1}\left(\cos\frac{\theta}{3} - i\sin\frac{\theta}{3}\right)\right) \quad (7.271)$$

$$t_{cp_{1,3}} = \frac{-1}{2}\left(2\sqrt{-Q_1}\cos\frac{\theta}{3}\right) - \frac{D_1}{4E_1} - \frac{1}{2}i\sqrt{3}\left(2i\sqrt{-Q_1}\sin\frac{\theta}{3}\right) \quad (7.272)$$

$$t_{cp_{1,3}} = -\sqrt{-Q_1}\cos\frac{\theta}{3} - \frac{D_1}{4E_1} + \sqrt{3}\sqrt{-Q_1}\sin\frac{\theta}{3} \quad (7.273)$$

$$t_{cp_{1,3}} = \sqrt{-Q_1}\left(-\cos\frac{\theta}{3} + \sqrt{3}\sin\frac{\theta}{3}\right) - \frac{D_1}{4E_1} \quad (7.274)$$

The set of equations obtained for the ending polynomial segment parallels that generated for the first.

2. Numerical Example. The times of zero velocity for the various spline polynomials of joint variable 1 are determined here. Note that as the θ_1 rotation has no physical limitations, the tests are not strictly required in this case. However, as this is not the case for other of the manipulator joints, the demonstration of the process will prove useful for them. Additionally, these tests serve to illustrate one aspect of the behavior of the spline polynomials.

a. Internal Spline Segments. Beginning with the spline for segment 2 in Equation (7.181), Equation (7.228) is first used to evaluate the discriminant for the solution set.

$$P_2(t) = 10 + (0.224)t + (1.866\text{E-}3)t^2 + (-1.965\text{E-}5)t^3 \tag{7.275}$$

$$d_2 = C_2^2 - 3B_2D_2 \tag{7.276}$$

$$d_2 = (1.866\text{E-}3)^2 - 3(0.224)(-1.965\text{E-}5) = 1.669\text{E-}5 \tag{7.277}$$

As the discriminant is positive, two real solutions, given by Equation (7.228), exist.

$$t_{cp_2} = \frac{-C_2 \pm \sqrt{d_2}}{3D_2} \tag{7.278}$$

$$t_{cp_2} = \frac{-(1.866\text{E-}3) \pm \sqrt{1.669\text{E-}5}}{3(-1.965\text{E-}5)} \tag{7.279}$$

$$t_{cp_2} = -37.648 \quad \text{or} \quad 100.956 \tag{7.280}$$

As the first time is negative, it is of no interest here. However, the second time is positive and less than the 137.441 maximum established in Equation (7.153), so the trajectory should be evaluated here.

$$P_2(100.956) = 10 + 0.224(100.956) + (1.866\text{E-3})(100.956)^2 + (-1.965\text{E-5})(100.956)^3 \tag{7.281}$$

$$P_2(100.956) = 31.414^{\circ} \tag{7.282}$$

For spline segment three of Equation (7.187),

$$P_3(t) = 25 + (-0.377)t + (-6.265\text{E-3})t^2 + (8.503\text{E-5})t^3 \tag{7.283}$$

$$d_3 = C_3^2 - 3B_3D_3 \tag{7.284}$$

$$d_3 = (-6.265\text{E-3})^2 - 3(-0.377)(8.503\text{E-5}) = 1.354\text{E-4} \tag{7.285}$$

$$t_{cp_3} = \frac{-C_3 \pm \sqrt{d_3}}{3D_3} \tag{7.286}$$

$$t_{cp_3} = \frac{-(-6.265\text{E-3}) \pm \sqrt{1.354\text{E-4}}}{3(8.503\text{E-5})} \tag{7.287}$$

$$t_{cp_3} = 70.176 \quad \text{or} \quad -21.056 \tag{7.288}$$

Since the first time is larger than the amount of time spent on the segment, 53.245, and the second time is negative, no tests need be made.

Continuing with the fourth spline segment from Equation (7.193),

$$P_4(t) = 0 + (-0.321)t + (7.316\text{E-3})t^2 + (-2.852\text{E-5})t^3 \tag{7.289}$$

$$d_4 = C_4^2 - 3B_4D_4 \tag{7.290}$$

$$d_4 = (7.316\text{E-}3)^2 - 3(-0.321)(-2.852\text{E-}5) = 2.606\text{E-}5 \tag{7.291}$$

$$t_{cp_4} = \frac{-C_4 \pm \sqrt{d_4}}{3D_4} \tag{7.292}$$

$$t_{cp_4} = \frac{-(7.316\text{E-}3) \pm \sqrt{2.606\text{E-}5}}{3(-2.852\text{E-}5)} \tag{7.293}$$

$$t_{cp_4} = 25.844 \quad \text{or} \quad 145.171 \tag{7.294}$$

The second time is outside of the interval spent on the segment, 136.963 as stated by Equation (7.155), so only the position at the first time need be tested.

$$\begin{aligned} P_4(25.844) = (-0.321)(25.844) + (7.316\text{E-}3)(25.844)^2 \\ + (-2.852\text{E-}5)(25.844)^3 \end{aligned} \tag{7.295}$$

$$P_4(25.844) = -3.902^{*} \tag{7.296}$$

The examination of the intermediate segments concludes with the test for $P_5(t)$ of Equation (7.199).

$$P_5(t) = 20 + (0.078)t + (-4.394\text{E-}3)t^2 + (8.640\text{E-}6)t^3 \tag{7.297}$$

$$d_5 = C_5^2 - 3B_5D_5 \tag{7.298}$$

$$d_5 = (-4.394\text{E-}3)^2 - 3(0.078)(8.640\text{E-}6) = 1.729\text{E-}5 \tag{7.299}$$

$$t_{cp_5} = \frac{-C_5 \pm \sqrt{d_5}}{3D_5} \tag{7.300}$$

$$t_{cp_5} = \frac{-(-4.394\text{E-}3) \pm \sqrt{1.729\text{E-}5}}{3(8.640\text{E-}6)} \tag{7.301}$$

$$t_{cp_5} = 329.943 \quad \text{or} \quad 9.100 \tag{7.302}$$

The first time is outside of the amount spent on the interval, 206.211, so only the second is tested.

$$P_5(9.100) = 20 + (0.078)(9.100) + (-4.394\text{E-3})(9.100)^2 + (8.640\text{E-6})(9.100)^3 \tag{7.303}$$

$$P_5(9.100) = 20.352^{*} \tag{7.304}$$

b. <u>The Terminal Spline Segments</u>. The first spline polynomial was stated in Equation (7.209); the determination of the critical times for the exceedance of joint boundaries begins by application of Equations (7.241), (7.246), and (7.247).

$$P_1(t) = (-1.116\text{E-7})t^3 + (9.819\text{E-9})t^4 \tag{7.305}$$

$$Q_1 = \frac{1}{2E_1}\left(\frac{C_1}{3} - \frac{D_1^2}{8E_1}\right) \tag{7.306}$$

$$Q_1 = \frac{1}{2(9.819\text{E-9})}\left(\frac{0}{3} - \frac{(-1.116\text{E-7})^2}{8(9.819\text{E-9})}\right) = -8.074 \tag{7.307}$$

$$R_1 = \frac{1}{8E_1}\left(\frac{D_1}{2E_1}\left(C_1 - \frac{D_1^2}{4E_1}\right) - B_1\right) \tag{7.308}$$

$$R_1 = \frac{1}{8(9.819\text{E-9})}\left(\frac{-1.116\text{E-7}}{2(9.819\text{E-9})}\left(0 - \frac{(-1.116\text{E-7})^2}{4(9.819\text{E-9})}\right) - 0\right) = 22.941 \tag{7.309}$$

$$d_1 = Q_1^3 + R_1^2 \tag{7.310}$$

$$d_1 = (-8.074)^3 + (22.941)^2 = -5.035\text{E-2} \tag{7.311}$$

Since d_1 is negative, there are three times at which the velocity becomes zero along the polynomial defined by $P_1(t)$. They are obtained from Equations (7.258), (7.266), (7.270), and (7.274)

$$\theta = \tan^{-1} \frac{\sqrt{-d_1}}{R_1} \tag{7.312}$$

$$\theta = \tan^{-1}\left(\frac{\sqrt{-(-5.035\text{E-2})}}{22.941} \right) = 0.560^{\circ} \tag{7.313}$$

$$t_{cp_{1,1}} = 2\sqrt{-Q_1}\, \cos\frac{\theta}{3} - \frac{D_1}{4E_1} \tag{7.314}$$

$$t_{cp_{1,1}} = 2\sqrt{-(-8.074)}\, \cos\frac{0.560^{\circ}}{3} - \frac{-1.116\text{E-7}}{4(9.819\text{E-9})} = 8.524 \tag{7.315}$$

$$t_{cp_{1,2}} = -\sqrt{-Q_1}\left(\cos\frac{\theta}{3} + \sqrt{3}\,\sin\frac{\theta}{3} \right) - \frac{D_1}{4E_1} \tag{7.316}$$

$$t_{cp_{1,2}} = -\sqrt{-(-8.074)}\left(\cos\frac{0.560^{\circ}}{3} + \sqrt{3}\,\sin\frac{0.560^{\circ}}{3} \right) - \frac{-1.116\text{E-7}}{4(9.819\text{E-9})} \tag{7.317}$$

$$t_{cp_{1,2}} = -0.016 \tag{7.318}$$

$$t_{cp_{1,3}} = \sqrt{-Q_1}\left(-\cos\frac{\theta}{3} + \sqrt{3}\,\sin\frac{\theta}{3} \right) - \frac{D_1}{4E_1} \tag{7.319}$$

$$t_{cp_{1,3}} = \sqrt{-(-8.074)}\left(-\cos\frac{0.560^{\circ}}{3} + \sqrt{3}\,\sin\frac{0.560^{\circ}}{3} \right) - \frac{-1.116\text{E-7}}{4(9.819\text{E-9})} \tag{7.320}$$

$$t_{cp_{1,3}} = 0.016 \tag{7.321}$$

The trajectory position is checked at $t_{cp_{1,1}}$ and $t_{cp_{1,3}}$, the two non-negative times, each of which is less than the maximum of 181.552 stated in Equation (7.152).

$$P_1(8.524) = (-1.116\text{E-}7)(8.524)^3 + (9.819\text{E-}9)(8.524)^4 = -1.728\text{E-}5 \tag{7.322}$$

$$P_1(0.016) = (-1.116\text{E-}7)(0.016)^3 + (9.819\text{E-}9)(0.016)^4 = -4.565\text{E-}13 \tag{7.323}$$

The procedure for the ending trajectory spline, stated in Equation (7.221), follows that of the first spline segment. Equations analogous to (7.241), (7.246), and (7.247) are employed first.

$$P_6(t) = -75 + (-0.632)t + (9.504\text{E-}4)t^2 + (2.493\text{E-}5)t^3 + (-1.003\text{E-}7)t^4 \tag{7.324}$$

$$Q_6 = \frac{1}{2E_6}\left(\frac{C_6}{3} - \frac{D_6^2}{8E_6}\right) \tag{7.325}$$

$$Q_6 = \frac{1}{2(-1.003\text{E-}7)}\left(9.504\text{E-}\frac{4}{3} - \frac{(2.493\text{E-}5)^2}{8(-1.003\text{E-}7)}\right) = -5.440E + 3 \tag{7.326}$$

$$R_6 = \frac{1}{8E_6}\left(\frac{D_6}{2E_6}\left(C_6 - \frac{D_6^2}{4E_6}\right) - B_6\right) \tag{7.327}$$

$$R_6 = \frac{1}{8(-1.003\text{E-}7)}\left(\frac{2.493\text{E-}5}{2(-1.003\text{E-}7)}\left(9.504\text{E-}4 - \frac{(2.493\text{E-}5)^2}{4(-1.003\text{E-}7)}\right) - (-0.632)\right) \tag{7.328}$$

$$R_6 = -4.005E + 5 \tag{7.329}$$

$$d_6 = Q_6^3 + R_6^2 \tag{7.330}$$

$$d_6 = (-5.440E + 3)^3 + (-4.005E + 5)^2 = -5.889E + 8 \tag{7.331}$$

Since the discriminant is negative, there are three times for zero velocity of the final trajectory. Equations (7.258), (7.266), (7.270), and (7.274) yield these times.

$$\theta_6 = \tan^{-1} \frac{\sqrt{-d}}{R} \tag{7.332}$$

$$\theta_6 = \tan^{-1}\left(\frac{\sqrt{-(-5.889E+8)}}{-4.005E+5} \right) = 176.533^\circ \tag{7.333}$$

$$t_{cp_{6,1}} = 2\sqrt{-Q_6}\ \cos\frac{\theta_6}{3} - \frac{D_6}{4E_6} \tag{7.334}$$

$$t_{cp_{6,1}} = 2\sqrt{-(-5.440E+3)}\ \cos\frac{176.533^\circ}{3} - \frac{2.493\text{E-}5}{4(-1.003\text{E-}7)} = 138.456 \tag{7.335}$$

$$t_{cp_{6,2}} = -\sqrt{-Q_6}\left(\cos\frac{\theta_6}{3} + \sqrt{3}\ \sin\frac{\theta_6}{3} \right) - \frac{D_6}{4E_6} \tag{7.336}$$

$$\begin{aligned} t_{cp_{6,2}} = -\sqrt{-(-5.440E+3)}\Bigg(&\cos\frac{176.533^\circ}{3} \\ &+ \sqrt{3}\ \sin\frac{176.533^\circ}{3} \Bigg) - \frac{2.493\text{E-}5}{4(-1.003\text{E-}7)} \end{aligned} \tag{7.337}$$

$$t_{cp_{6,2}} = -85.344 \tag{7.338}$$

$$t_{cp_{6,3}} = \sqrt{-Q_6}\left(-\cos\frac{\theta_6}{3} + \sqrt{3}\ \sin\frac{\theta_6}{3} \right) - \frac{D_6}{4E_6} \tag{7.339}$$

$$\begin{aligned} t_{cp_{6,3}} = \sqrt{-(-5.440E+3)}\Bigg(&-\cos\frac{176.533^\circ}{3} \\ &+ \sqrt{3}\ \sin\frac{176.533^\circ}{3} \Bigg) - \frac{2.493\text{E-}5}{4(-1.003\text{E-}7)} \end{aligned} \tag{7.340}$$

$$t_{cp_{6,3}} = 133.303 \tag{7.341}$$

As $t_{cp6,1}$ exceeds the maximum time on the interval, 135.831 as stated in Equation (7.157), and $t_{cp6,2}$ is negative, only $t_{cp6,3}$ need be checked.

$$P_6(133.303) = -75 + (-0.632)(133.303) + (9.504E\text{-}4)(133.303)^2 + (2.493E\text{-}5)(133.303)^3 + (-1.003E\text{-}7)(133.303)^4 \qquad (7.342)$$

$$P_6(133.303) = -114.977 \qquad (7.343)$$

Figure 7.8 graphically illustrates the results of this section concerning the spline trajectory. The positions where the spline may take on extreme values are added to the spline positions known in the previous section. As before, the values used here are those obtained by computer. As will be seen, there are some significant differences between the results obtained in this section and those of the program example.

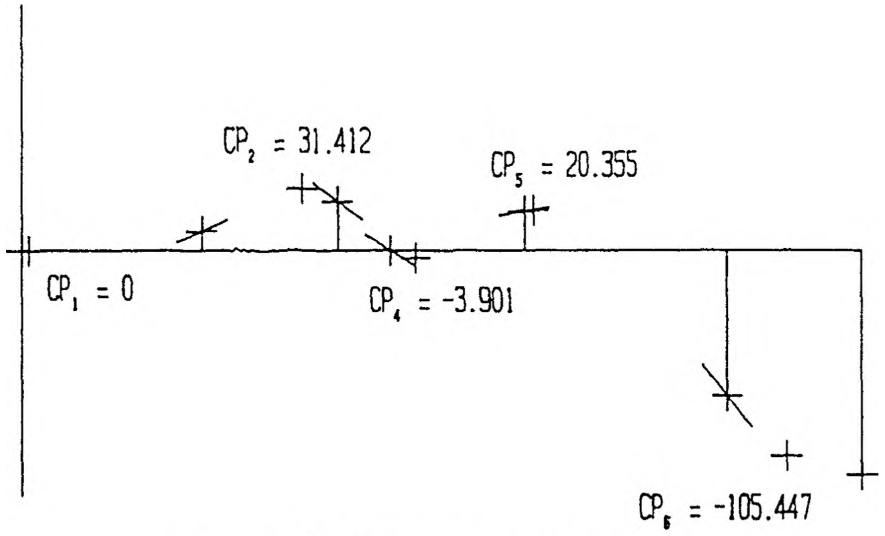


Figure 7.8. Trajectory Critical Positions

3. Program Control. Procedure **critical_positions** determines the times and corresponding positions for critical position points along each trajectory in order.

```
dsply_crit_pos (&row, cols);
terminal_crit_pos (1, a[1], b[1], c[1], d[1], e[1], t[1], lb, hb,
                   row, cols);

for (i = 2; i <= n-2; i++)
   intermediate_crit_pos (i, a[i], b[i], c[i], d[i], t[i], lb, hb,
                          row+i-1, cols);

terminal_crit_pos (n-1, a[n-1], b[n-1], c[n-1], d[n-1], e[n-1],
                     t[n-1], lb, hb, row+n-2, cols);
   wait_then_erase (10);
```

Procedure `terminal_crit_pos` determines critical times as the roots of a third degree equation, evaluating the positions at only the real, in-bounds times.

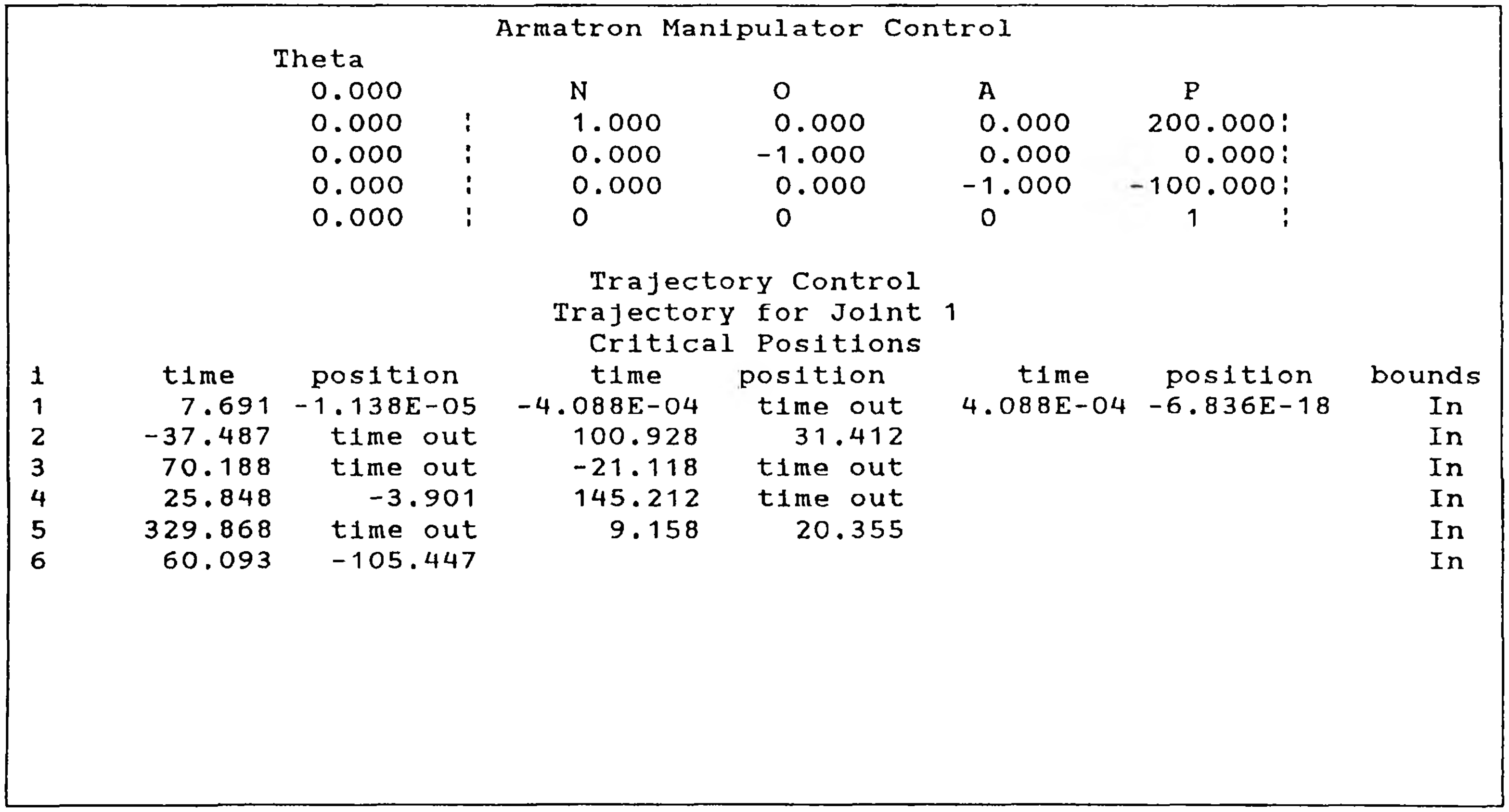
```
   locate (row, cols[0]);
   cprintf ("%d", i);
   q = (1 / (2 * e)) * ((c / 3) - square (d) / (8 * e));
   r = (1 / (8 * e)) * ((d / (2*e)) * (c - square (d) / (4 * e)) - b);
   discr = pow (q,3) + square (r);

   if ( discr > small_tolerance )
         {
         roots = 1;
         tcp[0] = cube_root (r + sqrt(discr))
                + cube_root (r - sqrt(discr)) - d / (4 * e);
         }
      else
         if ( fabs(discr) < small_tolerance )
               {
               roots = 2;
               tcp[0] = 2*cube_root (r) - d / (4 * e);
               tcp[1] =  -cube_root (r) - d / (4 * e);
               }
            else
               {
               roots = 3;
               x = atan2(sqrt(-discr), r);
               tcp[0] = 2 * sqrt(-q) * cos(x/3) - d/(4*e);
               tcp[1] = -sqrt(-q) * ( cos(x/3) + sqrt(3)*sin(x/3))
                      - d/(4*e);
               tcp[2] =  sqrt(-q) * (-cos(x/3) + sqrt(3)*sin(x/3))
                      - d/(4*e);
               }
   for (j = 0; j <= roots-1; j++)
      cp[j] = eval_cp (a, b, c, d, e, tcp[j], t,
                       row, cols[j*2+1], cols[j*2+2]);
   for (j = 0; j <= roots-1; j++)
      check_range (lb, hb, cp[j], row, cols[7]);
```

Procedure `intermediate_crit_pos` has the simpler task of determining critical times as roots of a quadratic equation.

```
   locate (row, cols[0]);
   cprintf ("%d", i);
   discr = square (c) - 3*b*d;
   if (discr > small_tolerance)
         {
         roots = 2;
         tcp[0] = (-c + sqrt(discr)) / (3 * d);
         tcp[1] = (-c - sqrt(discr)) / (3 * d);
         }
      else
         if ( fabs(discr) < small_tolerance )
               {
               roots = 1;
               tcp[0] = -c / (3 * d);
               }
            else
               roots = 0;
   for (j = 0; j <= roots-1; j++)
      cp[j] = eval_cp (a, b, c, d, 0, tcp[j], t,
                       row, cols[j*2+1], cols[j*2+2]);
   for (j = 0; j <= roots-1; j++)
```

```
check_range (lb, hb, cp[j], row, cols[7]);
```

4. Program Example. Figure 7.9 gives the display produced by procedure `critical_positions`. Comparison with the critical times of the terminal spline segments in particular will show that there is a significant loss of accuracy in the numerical example. Indeed, the discriminant for the final quartic trajectory segment turned out to be positive, as its components, Q_6^3 and R_6^2, while opposite in sign, were rather close in magnitude. The precision lost in the numerical example was sufficient to result in an effectively meaningless value for the discriminant.



```
                         Armatron Manipulator Control
              Theta
                0.000            N          O          A          P
                0.000   |      1.000      0.000      0.000    200.000|
                0.000   |      0.000     -1.000      0.000      0.000|
                0.000   |      0.000      0.000     -1.000   -100.000|
                0.000   |      0                    0          0          1      |

                              Trajectory Control
                            Trajectory for Joint 1
                              Critical Positions
i       time     position       time     position       time     position   bounds
1       7.691  -1.138E-05  -4.088E-04   time out   4.088E-04  -6.836E-18      In
2     -37.487    time out     100.928     31.412                              In
3      70.188    time out     -21.118   time out                              In
4      25.848      -3.901     145.212   time out                              In
5     329.868    time out       9.158     20.355                              In
6      60.093    -105.447                                                     In
```

Figure 7.9. Determination of Critical Positions

C. SCALING FOR OPTIMUM TRAJECTORY TRAVERSAL TIME

The extreme velocity and acceleration obtained along each spline must now be determined for the time units stated so that the physical limitations of the manipulator joints are not exceeded. Scaling factors will be obtained so that no maximum velocity or acceleration is exceeded. A final scaling factor will be obtained which will result in an optimum traversal of the trajectory.

1. Derivation of Equations. Consider the determination of the maximum velocity for each joint's segments. As long as the joint is accelerating or decelerating, the velocity is increasing or decreasing, respectively. Thus the velocity attains any maximum or minimum when the acceleration along the segment becomes zero. Note that as the spline polynomials were defined to be continuous in position, velocity, and acceleration, no additional examinations need be performed concerning the velocities at the spline nodes.

a. The Internal Segments. The maximum and minimum velocities attained along internal segments are considered first. The function of Equation (7.26), which defined the acceleration along an internal segment, is set equal to zero and solved for the critical time concerning the velocity, t_{cv_i}.

$$S''_i(t) = 2C_i + 6D_it \tag{7.344}$$

$$2C_i + 6D_it_{cv_i} = 0 \tag{7.345}$$

$$t_{cv_i} = \frac{-C_i}{3D_i} \tag{7.346}$$

The velocity of the joint as given by Equation (7.11) is then evaluated at this time as V_{c_i} for comparison with the maximum obtainable velocity of the joint.

$$S'_i(t) = B_i + 2C_i t + 3D_i t^2 \tag{7.347}$$

$$S'_i(t_{cv_i}) = B_i + 2C_i\left(\frac{-C_i}{3D_i}\right) + 3D_i\left(\frac{-C_i}{3D_i}\right)^2 \tag{7.348}$$

$$V_{c_i} = B_i - \frac{2C_i^2}{3D_i} + \frac{C_i^2 D_i}{3D_i^2} \tag{7.349}$$

$$V_{c_i} = B_i - \frac{C_i^2}{3D_i} \tag{7.350}$$

Of course, if t_{cv_i} falls outside of the time during which the joint will be traversing the spline, no maximum will be obtained along it.

The maximum and minimum accelerations obtained along the internal spline segments will occur when the acceleration is itself not increasing or decreasing. This occurs at the point when the derivative of the acceleration is zero. The acceleration function of Equation (7.26) is differentiated to show that the rate of change of the acceleration along an internal segment is constant.

$$S''_i(t) = 2C_i + 6D_i t \tag{7.351}$$

$$S'''_i(t) = 6D_i \tag{7.352}$$

If D_i is positive, the acceleration is increasing throughout the span of the segment and thus the acceleration takes on a minimum at $t = 0$ and a maximum at $t = t_i$. Conversely, if D_i is negative, the acceleration continues to decrease along the segment from a maximum at $t = 0$ to a minimum at $t = t_i$. Thus the acceleration needs to be tested only at the beginning of each segment.

$$S''_i(0) = 2C_i + 6D_i(0) \tag{7.353}$$

$$A_{c_i} = 2C_i \tag{7.354}$$

The continuity imposed upon the trajectory through velocity and acceleration will lead the accelerations at nodes 2 and $n - 1$ to be covered by the acceleration tests for the terminal spline segments.

b. The Terminal Segments. As was the case for the intermediate segments, the velocity along the first segment will take on an extreme value when the acceleration along the segment is zero. The difference here is that the acceleration function as defined by Equation (7.26) is quadratic, and thus may have two solutions; one of these solutions will be the unattached end of the spline for which both the velocity and acceleration were defined to be zero.

$$S''_1(t) = 2C_1 + 6D_1t + 12E_1t^2 \tag{7.355}$$

$$12E_1t_{cv_1}^2 + 6D_1t_{cv_1} + 2C_1 = 0 \tag{7.356}$$

$$6E_1t_{cv_1}^2 + 3D_1t_{cv_1} + C_1 = 0 \tag{7.357}$$

$$t_{cv_1} = \frac{-3D_1 \pm \sqrt{9D_1^2 - 4(6E_1)(C_1)}}{2(6E_1)} \tag{7.358}$$

$$t_{cv_1} = \frac{-D_1 \pm \sqrt{D_1^2 - (8C_1E_1/3)}}{4E_1} \tag{7.359}$$

The quantity $D_1^2 - (8C_1E_1/3)$, or d_1, may be positive, zero, or negative. Correspondingly, there will be either two, one, or zero times at which the velocity, as defined by Equation (7.50), may take on an extremum. As the terminal spline polynomials were defined such that the velocities at nodes 0 and n would be zero, one value for t_{cv_1} in Equation (7.359) must be zero. Let t_{cv_1} be the time not associated with an end of the segment. Due to the complexity of the formula for t_{cv_1}, the velocity is most efficiently determined by a straightforward evaluation of the velocity polynomial.

$$V_{c_1} = B_1 + 2C_1(t_{cv_1}) + 3D_1(t_{cv_1})^2 + 4E_1(t_{cv_1})^3 \qquad (7.360)$$

As with the internal segments, if t_{cv_1} does not occur during the time interval of segment 1, then the maximum velocity at that point is not a concern.

The acceleration function of Equation (7.26) is differentiated and set equal to zero in order to determine the time at which the acceleration along the first segment may reach an extremum.

$$S''_1(t) = 2C_1 + 6D_1t + 12E_1t^2 \qquad (7.361)$$

$$S'''_1(t) = 6D_1 + 24E_1t \qquad (7.362)$$

$$6D_1 + 24E_1t_{ca_1} = 0 \qquad (7.363)$$

$$t_{ca_1} = \frac{-D_1}{4E_1} \qquad (7.364)$$

The critical acceleration at this point is termed A_{c_1}.

$$S''_1(t_{ca_1}) = 2C_1 + 6D_1\left(\frac{-D_1}{4E_1}\right) + 12E_1\left(\frac{-D_1}{4E_1}\right)^2 \qquad (7.365)$$

$$A_{c_1} = 2C_1 - \frac{6D_1^2}{4E_1} + \frac{12D_1^2E_1}{16E_1} \qquad (7.366)$$

$$A_{c_1} = 2C_1 - \frac{3D_1^2}{4E_1} \qquad (7.367)$$

The equations for the critical velocities and acceleration along the final spline segment are identical in form to those along the first segment.

$$t_{cv_{n-1}} = \frac{-D_{n-1} \pm \sqrt{D_{n-1}^2 - (8C_{n-1}E_{n-1}/3)}}{4E_{n-1}} \qquad (7.368)$$

$$V_{c_{n-1}} = B_{n-1} + 2C_{n-1}(t_{cv_{n-1}}) + 3D_{n-1}(t_{cv_{n-1}})^2 + 4E_{n-1}(t_{cv_{n-1}})^3 \qquad (7.369)$$

$$t_{ca_{n-1}} = \frac{-D_{n-1}}{4E_1} \qquad (7.370)$$

$$A_{c_{n-1}} = 2C_{n-1} - \frac{3D_{n-1}^2}{4E_{n-1}} \qquad (7.371)$$

As with the other cases, the critical velocities and acceleration are of concern only if they occur during the time interval for which the manipulator joint is traversing the spline segment.

The extreme attained velocities of Equations (7.350), (7.360), and (7.369) and the extreme attained accelerations of Equations (7.354), (7.367), and (7.371) were derived in terms of the arbitrary time unit based upon the "distances" between the joint variable settings. These derived extrema must now be reconciled with the physical constraints under which the joints actually operate. The following rates are defined for this purpose.

$$V_m = \max\{|V_{c_i}|, \quad 1 \leq i \leq n-1\} \tag{7.372}$$

$$A_m = \max\{|A_{c_i}|, \quad 1 \leq i \leq n-1\} \tag{7.373}$$

If the maximum velocity magnitude attained on every spline traversed by the joint in question is less than the maximum rate the joint is capable of, then the path is not being traversed as quickly as possible. Conversely, if the maximum is greater than that attainable by the joint, the smooth motion defined by the trajectory cannot be achieved. To optimize this situation, a scale factor F is introduced to relate the "distance"-based time units of t to actual time T.

$$t = FT \tag{7.374}$$

$$T = \frac{t}{F} \tag{7.375}$$

Real time can be seen to be a fractional rate of the arbitrary time used thus far. If the manipulator is moving more slowly than need be with respect to t, F is given a value greater than one; the rates specified by the spline polynomials may be reduced by giving F a value less than one. The position function for an internal spline segment, given in Equation (7.6), is modified to reflect this. Note that the derivations of this section are not dependent upon the degree of the spline polynomial.

$$S_i(t) = A_i + B_it + C_it^2 + D_it^3 \tag{7.376}$$

$$S_i(T) = A_i + B_i(FT) + C_i(FT)^2 + D_i(FT)^3 \tag{7.377}$$

$$S_i(T) = A_i + B_iFT + C_iF^2T^2 + D_iF^3T^3 \tag{7.378}$$

The velocity function then becomes

$$S'_i(T) = B_iF + 2C_iF^2T + 3D_iF^3T^2 \tag{7.379}$$

Let T be replaced by Equation (7.375).

$$S'_i(T) = B_iF + 2C_iF^2\left(\frac{t}{F}\right) + 3D_iF^3\left(\frac{t}{F}\right)^2 \tag{7.380}$$

$$S'_i(T) = F(B_i + 2C_it + 3D_it^2) \tag{7.381}$$

$$S'_i(T) = FS'_i(t) \tag{7.382}$$

If the manipulator joint rates with respect to t call for the joint to be moving more rapidly than the velocity constraints will allow, factor F is given a value less than one. On the other hand, an F factor larger than one will increase the realized velocity. Under these criteria, F should be chosen to relate the maximum obtainable velocity for the joint, θ_{mv}, to the maximum rate determined for the trajectory with respect to t, V_m. The factor is labeled to reflect its dependence upon velocity constraints.

$$F_v = \frac{\theta_{mv}}{V_m} \tag{7.383}$$

This factor would be multiplied by the trajectory parameter t of each spline segment to either increase or decrease the rate at which the trajectory is being traversed. The result of this scaling is to call upon the joint for its maximum rate of change, but no more, when the trajectory reaches its point of extreme velocity.

Consider now the acceleration function with respect to T as obtained from the differentiation of the velocity function in Equation (7.379).

$$S''_i(T) = 2C_iF^2 + 6D_iF^3T \tag{7.384}$$

Variable T is again replaced by Equation (7.375).

$$S''_i(T) = 2C_iF^2 + 6D_iF^3\left(\frac{t}{F}\right) \tag{7.385}$$

$$S''_i(T) = F^2(2C_i + 6D_it) \tag{7.386}$$

$$S''_i(T) = F^2S''_i(t) \tag{7.387}$$

As was described for the examination of the velocity relationship, F should take on a value less than one if the joint acceleration constraint θ_{ma} is exceeded or more than one if the acceleration required is less than what the joint is capable of. The difference here is that the factor when dependent upon acceleration criteria is squared. The maximum acceleration with respect to t specified by the trajectory was seen to be A_m.

$$F_A^2 = \frac{\theta_{ma}}{A_m} \tag{7.388}$$

When factor F_a is multiplied by the trajectory parameter t of each spline segment, the rate of change of the velocity of the joint is either increased or decreased so that the maximum acceleration requested of the joint is precisely that which it can provide.

Factors F_v and F_a must be further resolved with one another by selecting the smaller of the two as F, the overall trajectory factor.

$$F = \min\{F_a, \ F_v\} \tag{7.389}$$

Thus only the maximum velocity or the maximum acceleration will be utilized, but not both. This is the optimal solution for traversal of the path designated by the spline trajectory.

To this point, only one joint trajectory has been considered. In the case of the Armatron robot arm, a five-degrees-of-freedom manipulator, five different trajectories will be generated around the same set of "distances" between path nodes. Each joint trajectory will however pass through a different set of node values, consequently specifying different velocity and acceleration extrema for each. Equations (7.372) and (7.373) are restated to take the different trajectories into account.

$$V_{m_j} = \max\{\,|V_{c_{i,j}}|, \quad 1 \leq i \leq n-1\} \tag{7.390}$$

$$A_{m_j} = \max\{\,|A_{c_{i,j}}|, \quad 1 \leq i \leq n-1\} \tag{7.391}$$

Terms $V_{c_{i,j}}$ and $A_{c_{i,j}}$ are now the maximum velocity and acceleration, respectively, on segment i of trajectory j. Scale factors are then determined for each trajectory.

$$F_{v_j} = \frac{\theta_{mv_j}}{V_{m_j}} \tag{7.392}$$

$$F_{a_j}^2 = \frac{\theta_{ma_j}}{A_{m_j}} \tag{7.393}$$

The smaller of the two factors is chosen for each joint so as not to exceed the maximum velocity or acceleration attainable by joint j.

$$F_j = \min\{F_{v_j},\ F_{a_j}\} \tag{7.394}$$

Finally, the overall scaling factor F is chosen as the minimum factor over all five joint trajectories so as not to exceed any joint velocity or acceleration maximum.

$$F = \min\{F_j, \quad 1 \leq j \leq 5\} \tag{7.395}$$

The "distances" between trajectory nodes are now divided by the scaling factor to yield the actual amount of time T_i involved in traversing each segment.

$$T_i = \frac{t_i}{F} \tag{7.396}$$

The total time required to traverse the trajectory is the sum of the individual segment times.

(7.397)

Determination of the joint positioning, velocity, and acceleration of the manipulator at any elapsed real time T_R, $0 \leq T_R \leq T_T$, begins by determining the spline segment which is in effect at time T_R. This is done by subtracting segment times from T_R until the next subtraction would result in a negative value.

$$T = T_R - \sum_{k=1}^{i-1} T_k \quad \ni \quad T_R - \sum_{k=1}^{i} T_k \leq 0 \leq T_R - \sum_{k=1}^{i-1} T_k \tag{7.398}$$

Time T is then multiplied by the scale factor before substitution into the appropriate segment i functions.

$$t = TF \tag{7.399}$$

$$S_i(t) = A_i + B_i t + C_i t^2 + D_i t^3 \;\; [+ E_i t^4] \tag{7.400}$$

$$S'_i(t) = B_i + 2C_i t + 3D_i t^2 \;\; [+ 4E_i t^3] \tag{7.401}$$

$$S''_i(t) = 2C_i + 6D_i t \;\; [+ 12E_i t^2] \tag{7.402}$$

The last term in each equation is employed for values of i of 1 or $n-1$.

2. Numerical Example. The extreme velocities obtained along the internal segments are obtained first using Equation (7.346) and (7.350) for the critical time and corresponding velocity, respectively.

$$S_2(t) = 10 + (0.224)t + (1.866\text{E-}3)t^2 + (-1.965\text{E-}5)t^3 \tag{7.403}$$

$$t_{cv_2} = \frac{-C_2}{3D_2} \tag{7.404}$$

$$t_{cv_2} = -1.866\text{E-}\frac{3}{3(-1.965\text{E-}5)} = 31.654 \tag{7.405}$$

The time spent on the second interval is 137.441 from Equation (7.153), so the velocity must be evaluated at t_{cv_2}.

$$V_{c_2} = B_2 - \frac{C_2^2}{3D_2} \tag{7.406}$$

$$V_{c_2} = 0.224 - \frac{(1.866\text{E-}3)^2}{3(-1.965\text{E-}5)} = 0.283 \tag{7.407}$$

The extreme acceleration along the interval is taken to be that at the beginning of the interval as the acceleration's rate of change was seen to be constant in Equation (7.352).

$$A_{c_2} = 2C_2 \tag{7.408}$$

$$A_{c_2} = 2(1.866\text{E}-3) = 3.732\text{E-}3 \tag{7.409}$$

The third segment's spline polynomial was stated in Equation (7.187).

$$S_3(t) = 25 + (-0.377)t + (-6.265\text{E-}3)t^2 + (8.503\text{E-}5)t^3 \tag{7.410}$$

$$t_{cv_3} = \frac{-C_3}{3D_3} \tag{7.411}$$

$$t_{cv_3} = -\frac{(-6.265\text{E-}3)}{3(8.503\text{E-}5)} = 24.560 \tag{7.412}$$

From Equation (7.154), the time spent on the third interval is 53.245, so an examination of the velocity is required at t_{cv_3}.

$$V_{c_3} = B_3 - \frac{C_3^2}{3D_3} \tag{7.413}$$

$$V_{c_3} = -0.377 - \frac{(-6.265\text{E-}3)^2}{3(8.503\text{E-}5)} = -0.531 \tag{7.414}$$

The critical acceleration on the third segment is

$$A_{c_3} = 2C_3 \tag{7.415}$$

$$A_{c_3} = 2(-6.265\text{E}-3) = -1.253\text{E-}2 \tag{7.416}$$

Equation (7.193) gives the spline polynomial of the fourth segment.

$$S_4(t) = 0 + (-0.321)t + (7.316\text{E-}3)t^2 + (-2.852\text{E-}5)t^3 \tag{7.417}$$

$$t_{cv_4} = \frac{-C_4}{3D_4} \tag{7.418}$$

$$t_{cv_4} = -\frac{(7.316\text{E-}3)}{3(-2.852\text{E-}5)} = 85.507 \tag{7.419}$$

The velocity must be evaluated at t_{cv_4} as the fourth interval is in use for 136.963 units, as specified by Equation (7.155).

$$V_{c_4} = B_4 - \frac{C_4^2}{3D_4} \tag{7.420}$$

$$V_{c_4} = -0.321 - \frac{(7.316\text{E-}3)^2}{3(-2.852\text{E-}5)} = 0.305 \tag{7.421}$$

The fourth segment's critical acceleration is

$$A_{c_4} = 2C_4 \tag{7.422}$$

$$A_{c_4} = 2(7.316\text{E}-3) = 1.463\text{E-}2 \tag{7.423}$$

The last internal spline polynomial, that of the fifth segment, was given in Equation (7.199).

$$P_5(t) = 20 + (0.078)t + (-4.394\text{E-}3)t^2 + (8.640\text{E-}6)t^3 \tag{7.424}$$

$$t_{cv_5} = \frac{-C_5}{3D_5} \tag{7.425}$$

$$t_{cv_5} = -\frac{(-4.394\text{E-}3)}{3(8.640\text{E-}6)} = 169.522 \tag{7.426}$$

Equation (7.156) states that the time on the fifth interval is 206.211, so the velocity must be evaluated at t_{cv_5}.

$$V_{c_5} = B_5 - \frac{C_5^2}{3D_5} \tag{7.427}$$

$$V_{c_5} = 0.078 - \frac{(-4.394\text{E-}3)^2}{3(8.640\text{E-}6)} = -0.667 \tag{7.428}$$

The critical acceleration on the fifth segment is

$$A_{c_5} = 2C_5 \tag{7.429}$$

$$A_{c_5} = 2(-4.394\text{E}-3) = -8.788\text{E-}3 \tag{7.430}$$

For the first spline segment, the critical times were expressed by Equation (7.359); the polynomial was stated in Equation (7.209).

$$S_1(t) = (-1.116\text{E-}7)t^3 + (9.819\text{E-}9)t^4 \tag{7.431}$$

$$t_{cv_1} = \frac{-D_1 \pm \sqrt{D_1^2 - (8C_1E_1/3)}}{4E_1} \tag{7.432}$$

$$t_{cv_1} = \frac{-(-1.116\text{E-}7) \pm \sqrt{(-1.116\text{E-}7)^2 - 8(0)(9.819\text{E-}9)/3}}{4(9.819\text{E-}9)} \tag{7.433}$$

$$t_{cv_1} = 5.683 \quad \text{or} \quad 0 \tag{7.434}$$

The second time expressed for t_{cv_1} is associated with the zero acceleration condition of the beginning of the interval and is thus discarded. The velocity is determined for the remaining time as it is less than that spent on the first segment, 181.552 units in Equation (7.152); the velocity equation was given by Equation (7.360).

$$V_{c_1} = B_1 + 2C_1t_{cv_1} + 3D_1t_{cv_1}^2 + 4E_1t_{cv_1}^3 \tag{7.435}$$

$$V_{c_1} = 0 + 2(0)(5.683) + 3(-1.116\text{E-}7)(5.683)^2 + 4(9.819\text{E-}9)(5.683)^3 \tag{7.436}$$

$$V_{c_1} = -3.604\text{E-}6 \tag{7.437}$$

The time and value of the extreme acceleration is found using Equations (7.364) and (7.367), respectively.

$$t_{ca_1} = \frac{-D_1}{4E_1} \tag{7.438}$$

$$t_{ca_1} = \frac{-(-1.116\text{E-}7)}{4(9.819\text{E-}9)} = 2.841 \tag{7.439}$$

Like the time of the critical velocity above, this time is also on the interval of the first segment.

$$A_{c_1} = 2C_1 - \frac{3D_1^2}{4E_1} \tag{7.440}$$

$$A_{c_1} = 2(0) - \frac{3(-1.116\text{E-}7)^2}{4(9.819\text{E-}9)} = -9.513\text{E-}7 \tag{7.441}$$

The polynomial of the final segment in Equation (7.221) has its times of critical velocity determined using Equation (7.368).

$$P_6(t) = -75 + (-0.632)t + (9.504\text{E-}4)t^2 + (2.493\text{E-}5)t^3 + (-1.003\text{E-}7)t^4 \tag{7.442}$$

$$t_{cv_6} = \frac{-D_6 \pm \sqrt{D_6^2 - (8C_6E_6/3)}}{4E_6} \tag{7.443}$$

$$t_{cv_6} = \frac{-(2.493\text{E-}5) \pm \sqrt{(2.493\text{E-}5)^2 - (8(9.504\text{E-}4)(-1.003\text{E-}7)/3)}}{4(-1.003\text{E-}7)} \tag{7.444}$$

$$t_{cv_6} = -11.621 \quad \text{or} \quad 135.898 \tag{7.445}$$

The first time is negative and consequently of no importance here. The second solution is the ending time of the interval, as a comparison with Equation (7.157) verifies; both the acceleration and the velocity are defined to be zero here. Thus, no critical velocities exist for the final segment. Equations (7.370) and (7.371) are employed to detect time of occurrence and value, respectively, of the extreme acceleration on the segment.

$$t_{ca_6} = \frac{-D_6}{4E_6} \tag{7.446}$$

$$t_{ca_6} = \frac{-(2.493\text{E-}5)}{4(-1.003\text{E-}7)} = 62.139 \tag{7.447}$$

This time is within the interval spent on the last segment, unlike the time of the critcal velocity.

$$A_{c_6} = 2C_6 - \frac{3D_6^2}{4E_6} \tag{7.448}$$

$$A_{c_6} = 2(9.504\text{E-}4) - \frac{3(2.493\text{E-}5)^2}{4(-1.003\text{E-}7)} = 6.548\text{E-}3 \tag{7.449}$$

Equations (7.390) and (7.391) are now employed to determine the extreme velocity and acceleration, respectively, for the trajectory of the first joint.

$$V_{m_1} = \max\{|V_{c_{i,1}}|, \quad 1 \le i \le 6\} \tag{7.450}$$

$$V_{m_1} = \max\{|-3.604\text{E-}6|, |0.283|, |-0.531|, \\ |0.305|, |-0.667|, (none)\} \tag{7.451}$$

$$V_{m_1} = 0.667 \tag{7.452}$$

$$A_{m_1} = \max\{|A_{c_{i,1}}|, \quad 1 \le i \le 6\} \tag{7.453}$$

$$A_{m_1} = \max\{|-9.513\text{E-}7|, |3.732\text{E-}3|, |-1.253\text{E-}2|, \\ |1.463\text{E-}2|, |-8.788\text{E-}3|, |6.548\text{E-}3|\} \tag{7.454}$$

$$A_{m_1} = 1.463\text{E-2} \qquad (7.455)$$

The scaling factors associated with these extrema are given by Equations (7.392) and (7.393). The maximum joint velocity used is the constant velocity for the joint with the Armatron manipulator as configured; the acceleration is selected arbitrarily as the configuration does not provide for control of change in velocity.

$$\theta_{mv_1} = 60^\circ/\text{sec} \qquad (7.456)$$

$$F_{v_1} = \frac{\theta_{mv_1}}{V_{m_1}} \qquad (7.457)$$

$$F_{v_1} = \frac{60^\circ/\text{sec}}{0.667^\circ/\text{unit of t}} = 89.955 \qquad (7.458)$$

$$\theta_{ma_1} = 100^\circ/\text{sec}^2 \qquad (7.459)$$

$$F_{a_1}^2 = \frac{\theta_{ma_1}}{A_{m_1}} \qquad (7.460)$$

$$F_{a_1} = \sqrt{\frac{100^\circ/\text{sec}^2}{(1.463\text{E-2}^\circ)/\text{unit of t}^2}} = 82.676 \qquad (7.461)$$

The smaller factor is chosen for the trajectory of the first joint by Equation (7.394).

$$F_1 = \min\{F_{v_1},\ F_{a_1}\} \qquad (7.462)$$

$$F_1 = \min\{89.955,\ 82.676\} = 82.676 \qquad (7.463)$$

Looking ahead to the results to be obtained by computer for the remaining factors, Equation (7.395) may be used to determine the scaling factor to be used on each segment of each of the joint trajectories.

$$F = \min\{F_j, \quad 1 \leq j \leq 5\} \tag{7.464}$$

$$F = \min\{82.676, \quad 137.542, \quad 91.542, 89.071, \quad 44.482\} = 44.482 \tag{7.465}$$

The actual time spent on each trajectory segment is then obtained from Equation (7.396) and Equations (7.152) through (7.157).

$$T_i = \frac{t_i}{F} \tag{7.466}$$

$$T_1 = \frac{181.552}{44.482} = 4.081 \tag{7.467}$$

$$T_2 = \frac{137.441}{44.482} = 3.090 \tag{7.468}$$

$$T_3 = \frac{53.245}{44.482} = 1.197 \tag{7.469}$$

$$T_4 = \frac{136.963}{44.482} = 3.079 \tag{7.470}$$

$$T_5 = \frac{206.211}{44.482} = 4.636 \tag{7.471}$$

$$T_6 = \frac{135.831}{44.482} = 3.054 \tag{7.472}$$

The individual segment times are added by Equation (7.397) to find the total manipulator motion time.

$$T_T = \sum_{i=1}^{n-1} T_i \tag{7.473}$$

$$T_T = 4.081 + 3.090 + 1.197 + 3.079 + 4.636 + 3.054 \tag{7.474}$$

$$T_T = 19.137 \tag{7.475}$$

The position of each of the manipulator joint variables may now be determined along the desired trajectory at any real time T_R, $0 \leq T_R \leq T_T$. Equation (7.398) specified the determination of the segment and elapsed real time on that segment.

$$T = T_R - \sum_{k=1}^{i-1} T_k \quad \ni \quad T_R - \sum_{k=1}^{i} T_k \leq 0 \leq T_R - \sum_{k=1}^{i-1} T_k \tag{7.476}$$

For example, consider a real time of 19 seconds. Equation (7.476) would require a value of i of 6 to be satisfied.

$$T = 19 - \sum_{k=1}^{5} T_k \quad \ni \quad 19 - \sum_{k=1}^{6} T_k \leq 0 \leq 19 - \sum_{k=1}^{5} T_k \tag{7.477}$$

$$T = 19 - 16.083 \quad \ni \quad 19 - 19.137 \leq 0 \leq 19 - 16.083 \tag{7.478}$$

$$T = 2.917 \tag{7.479}$$

The scale time for evaluation of the joint variables on segment 6 would then be found by multiplication of the segment real time and the scale factor, as specified by Equation (7.399).

$$t = TF \tag{7.480}$$

$$t = (2.917)(44.482) = 129.754 \tag{7.481}$$

The setting of the first joint variable is then found by application of the spline polynomial for segment 6 as given in Equation (7.221).

$$P_6(t) = -75 + (-0.632)t + (9.504E\text{-}4)t^2 + (2.493E\text{-}5)t^3 + (-1.003E\text{-}7)t^4 \tag{7.482}$$

$$P_6(129.754) = -114.973 \tag{7.483}$$

The settings of the four remaining joint variables are found by application of the segment 6 spline polynomials for the respective joints.

Figure 7.10 adds the velocity data obtained in this section to that of the previous sections. The velocity is the rate of change of the position, so the sign of the velocity indicates whether the spline function is increasing or decreasing. The values depicted for the critical velocities and the critical accelerations of the next figure are taken from the upcoming program example.

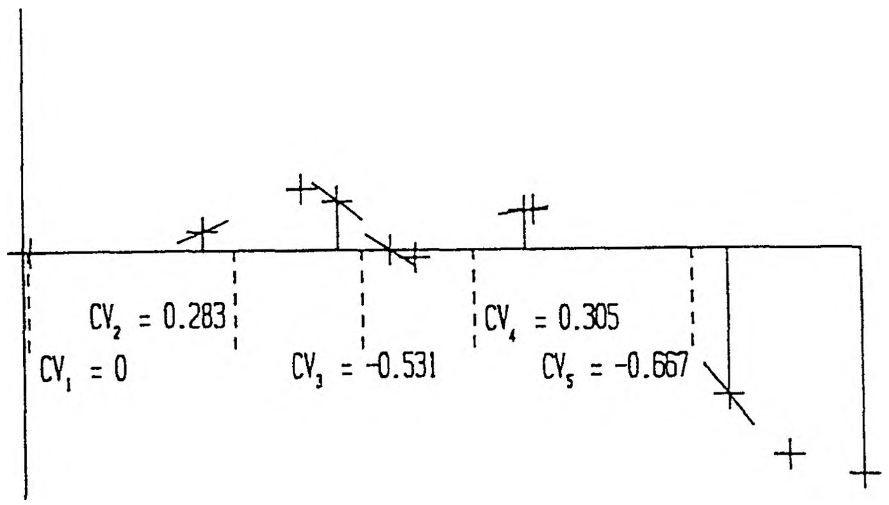


Figure 7.10. Trajectory Critical Velocities

Figure 7.11 completes the graph by combining the acceleration information with that of Figure 7.10. Since acceleration is the rate of change, or slope, of the velocity, a positive acceleration indicates that the trajectory spline is concave upward, as the slope of a line tangent to the curve must increase as time increases; conversely, a negative acceleration indicates a concave downward segment of the spline.

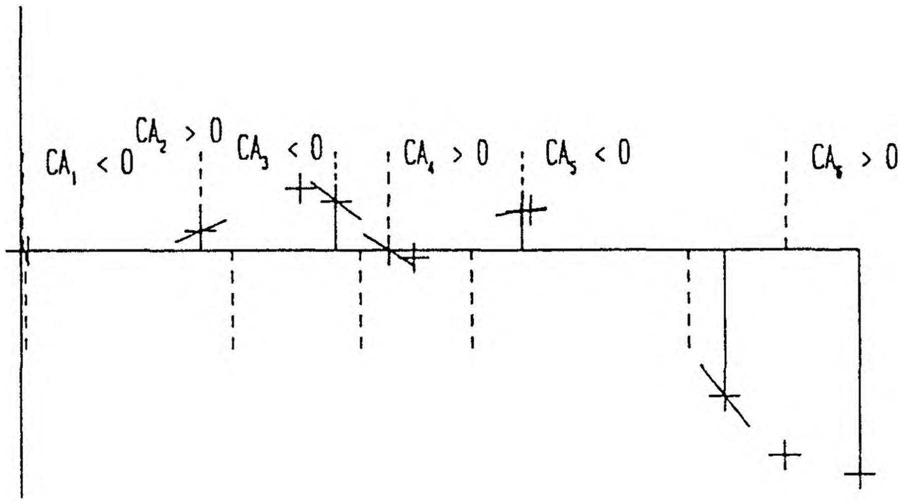


Figure 7.11. Trajectory Critical Accelerations

3. Program Control. Procedure `traj_scaling` governs the determination of critical velocities and accelerations on a trajectory for one of the manipulator joints.

```
dsply_traj_scaling (&row, cols);
maxv = terminal_crit_vel (1, b[1], c[1], d[1], e[1], t[1],
                         row, cols);
maxa = terminal_crit_acc (c[1], d[1], e[1], t[1], row, cols);
for (i = 2; i <= n-2; i++)
   {
   cv = intermediate_crit_vel (i, b[i], c[i], d[i], t[i],
                               row+i-1, cols);
   if ( cv > maxv )
      maxv = cv;
   ca = intermediate_crit_acc (c[i], row+i-1, cols);
   if ( ca > maxa )
      maxa = ca;
   }
cv = terminal_crit_vel (n-1, b[n-1], c[n-1], d[n-1], e[n-1], t[n-1],
                        row+n-2, cols);
if ( cv > maxv )
   maxv = cv;
ca = terminal_crit_acc (c[n-1], d[n-1], e[n-1], t[n-1],
                        row+n-2, cols);
if ( ca > maxa )
```

```
    maxa = ca;

lcprintf (row+6, cols[7], maxv);
lcprintf (row+7, cols[7], maxa);

fv = theta_maxv / maxv;
fa = sqrt(theta_maxa / maxa);
if ( fv < fa )
        f = fv;
    else
        f = fa;
lcprintf (row+8, cols[7], f);
wait_then_erase (9);
return (f);
```

Note that the maximum scaling factor for the trajectory under scrutiny is determined and returned by the procedure. Procedure `terminal_crit_vel` evaluates times of critical velocity as the roots of a second degree equation.

```
locate (row, cols[0]);
cprintf ("%2d", i);
discr = square (d) - 8*c*e/3;
if ( discr > small_tolerance)
        {
        roots = 2;
        tcv[0] = (-d + sqrt(discr)) / (4*e);
        tcv[1] = (-d - sqrt(discr)) / (4*e);
        }
    else
        if ( fabs(discr) < small_tolerance )
                {
                roots = 1;
                tcv[0] = -d / (4*e);
                }
            else
                roots = 0;
for (j = 0; j <= roots-1; j++)
    cv[j] = eval_cv (b, c, d, e, tcv[j], t,
                        row, cols[2*j+1], cols[2*j+2]);
maxv = 0;
for (j = 0; j <= roots-1; j++)
    if ( (mag = fabs(cv[j])) > maxv )
        maxv = mag;
return (maxv);
```

Procedure `intermediate_crit_vel` determines the single time of critical velocity as the root of a linear equation.

```
locate (row, cols[0]);
cprintf ("%2d  ", i);
tcv = -c / (3*d);
cv = eval_cv (b, c, d, 0, tcv, t, row, cols[1], cols[2]);
return (fabs(cv));
```

Procedure `terminal_crit_acc` likewise obtains a single time of critical acceleration as the root of a linear equation.

```
tca = -d / (4*e);
lcprintf (row, cols[5], tca);
if ( (tca >= 0) & (tca <= t) )
        {
```

```
        ca = 2*c - (3*square (d)) / (4*e);
        lcprintf (row, cols[6], ca);
        }
     else
        {
        ca = 0;
        lcputs (row, cols[6], " time out ");
        }
  return (fabs(ca));
```

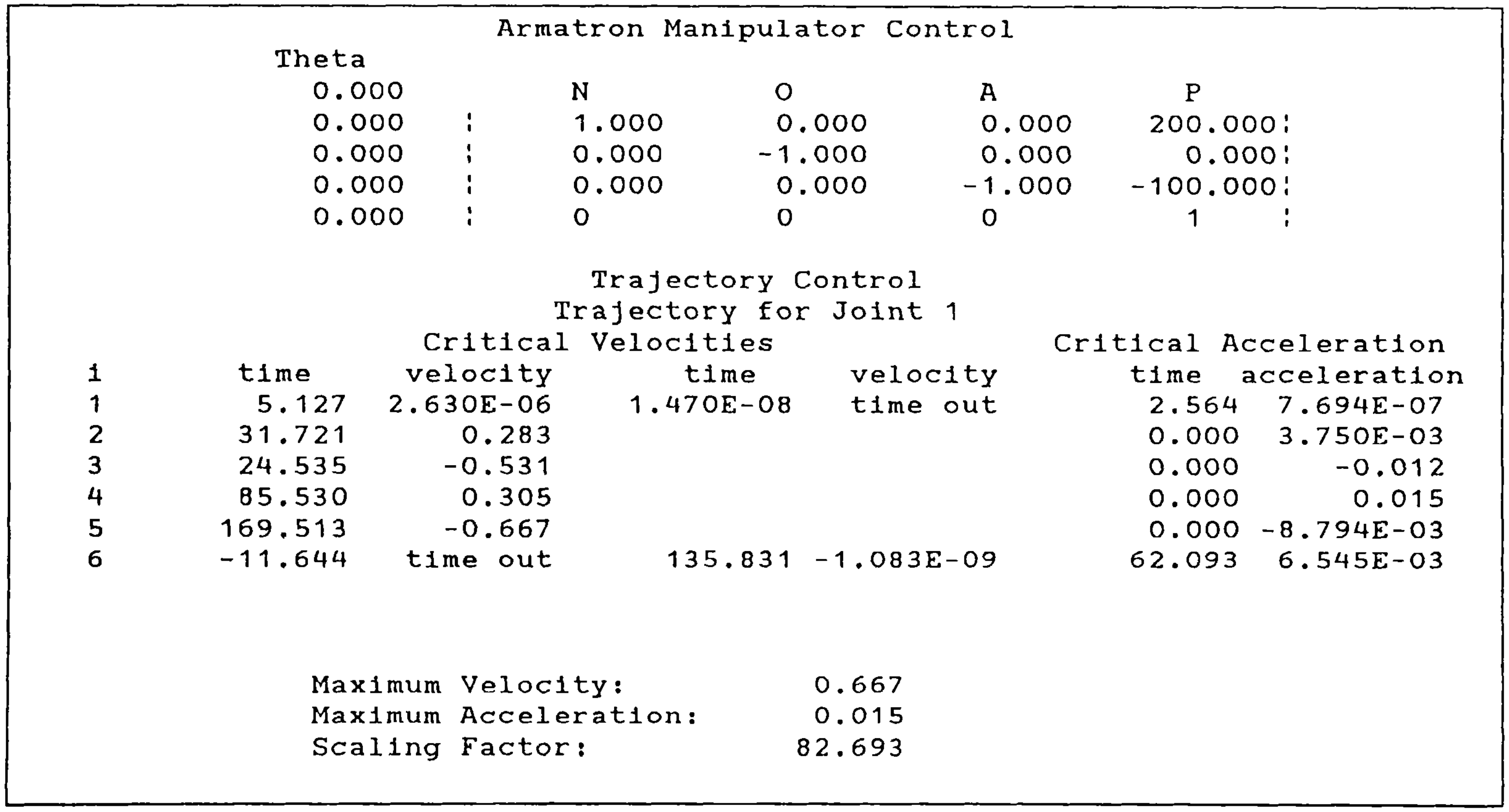
Procedure `intermediate_crit_acc` takes as critical time the starting point of the interval as the acceleration function is constant.

```
  ca = 2*c;
  lcprintf (row, cols[5], 0);
  lcprintf (row, cols[6], ca);
  return (fabs(ca));
```

Procedure `determine_positions` is invoked from the trajectory control procedure to allow for the evaluation of the spline polynomials at any real time. The cumulative time on the trajectory at each node is determined for ease of application in Equation (7.398).

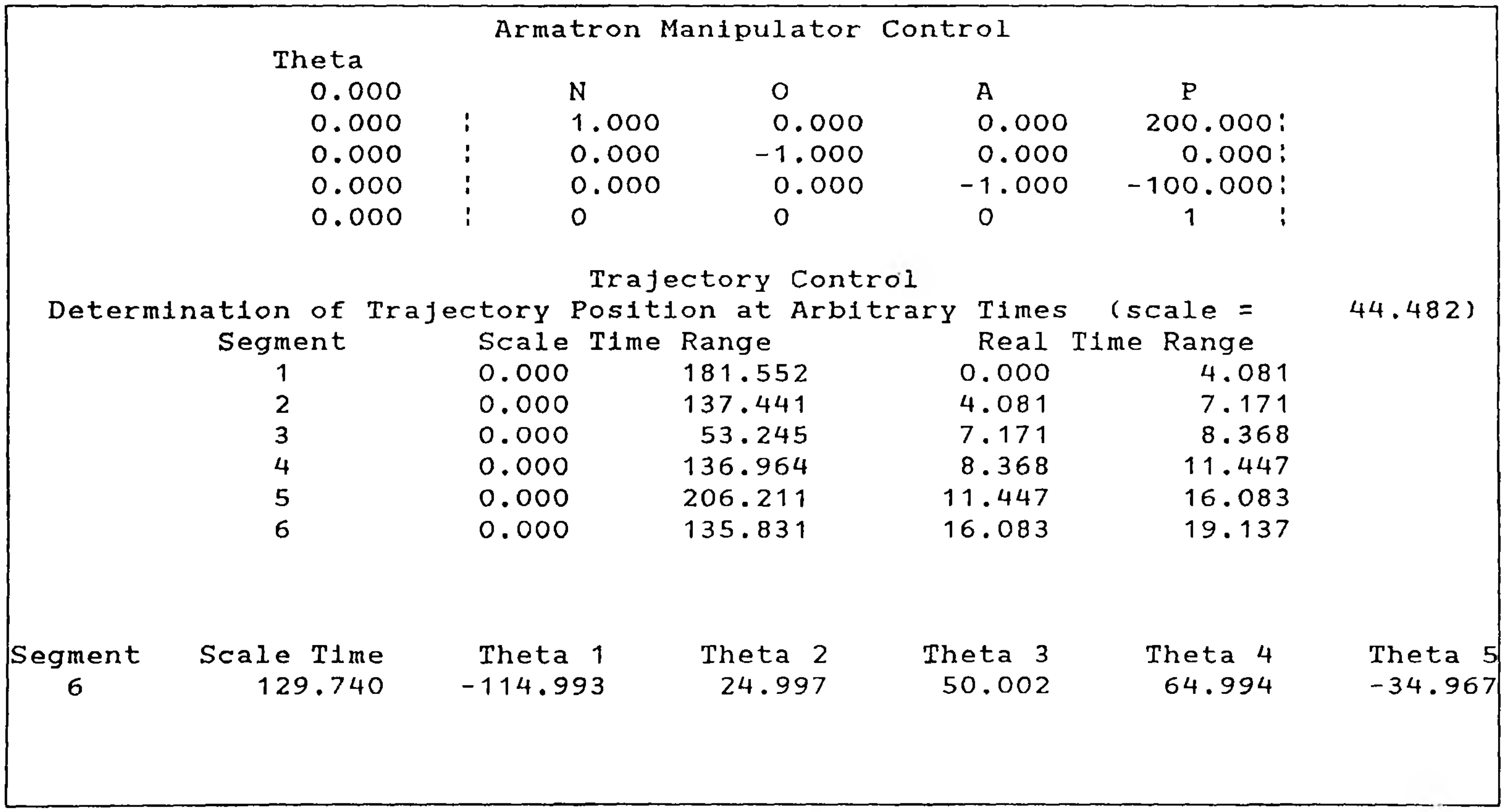
```
  dsply_det_positions (rows, time_cols, eval_cols, &scale_col);
  lcprintf (rows[2], scale_col, f);
  cumulative_time[0] = 0;
  for (i = 1; i <= n-1; i++)
     {
     locate (rows[0]+i-1, time_cols[0]);
     cprintf ("%d", i);
     lcprintf (rows[0]+i-1, time_cols[1], 0);
     lcprintf (rows[0]+i-1, time_cols[2], t[i]);

     cumulative_time[i] = cumulative_time[i-1] + t[i]/f;
     lcprintf (rows[0]+i-1, time_cols[3], cumulative_time[i-1]);
     lcprintf (rows[0]+i-1, time_cols[4], cumulative_time[i]);
     }

  while ( (real_time = get_time (cumulative_time, n)) != -1 )
     {
     i = 0;
     do
        i++;
        while ( real_time > cumulative_time[i] );
     locate (rows[1], eval_cols[0]);
     cprintf ("%d", i);
     scale_time = (real_time - cumulative_time[i-1]) * f;
     lcprintf (rows[1], eval_cols[6], scale_time);
     for (j = 1; j <= 5; j++)
        {
        trajectory_pos = a[j][i] + b[j][i] * scale_time
                       + c[j][i] * square (scale_time)
                       + d[j][i] * pow (scale_time, 3)
                       + e[j][i] * pow (scale_time, 4);
        lcprintf (rows[1], eval_cols[j], trajectory_pos);
        }
     }
  wait_then_erase (9);
  }
```

4. Program Example. The critical times and associated velocities and accelerations for the trajectory of the first manipulator joint are given by Figure 7.12. The values obtained show a lack of precision in the numerical example, but the results are essentially the same. Figure 7.13 shows the scale and real time ranges for the combined spline polynomials of the specified trajectory. Additionally, the results of an evaluation of the spline polynomials at real time = 19 seconds is given. Comparison of the θ_1 position with the result obtained in Equation (7.483) shows little difference.



```
                    Armatron Manipulator Control
             Theta
               0.000            N          O          A          P
               0.000    :     1.000      0.000      0.000     200.000:
               0.000    :     0.000     -1.000      0.000       0.000:
               0.000    :     0.000      0.000     -1.000    -100.000:
               0.000    :     0     0       0       1     :

                         Trajectory Control
                       Trajectory for Joint 1
                  Critical Velocities                 Critical Acceleration
  i        time       velocity       time      velocity        time  acceleration
  1         5.127   2.630E-06     1.470E-08    time out       2.564   7.694E-07
  2        31.721       0.283                                 0.000   3.750E-03
  3        24.535      -0.531                                 0.000      -0.012
  4        85.530       0.305                                 0.000       0.015
  5       169.513      -0.667                                 0.000  -8.794E-03
  6       -11.644    time out        135.831 -1.083E-09      62.093   6.545E-03



               Maximum Velocity:          0.667
               Maximum Acceleration:      0.015
               Scaling Factor:           82.693
```

Figure 7.12. Critical Velocities and Accelerations



```
                        Armatron Manipulator Control
              Theta
                0.000            N           O           A           P
                0.000   :      1.000       0.000       0.000     200.000:
                0.000   :      0.000      -1.000       0.000       0.000:
                0.000   :      0.000       0.000      -1.000    -100.000:
                0.000   :      0           0           0           1     :

                             Trajectory Control
  Determination of Trajectory Position at Arbitrary Times  (scale =      44.482)
           Segment         Scale Time Range             Real Time Range
              1            0.000       181.552         0.000          4.081
              2            0.000       137.441         4.081          7.171
              3            0.000        53.245         7.171          8.368
              4            0.000       136.964         8.368         11.447
              5            0.000       206.211        11.447         16.083
              6            0.000       135.831        16.083         19.137



Segment    Scale Time      Theta 1     Theta 2     Theta 3     Theta 4     Theta 5
   6          129.740    -114.993      24.997      50.002      64.994     -34.967
```

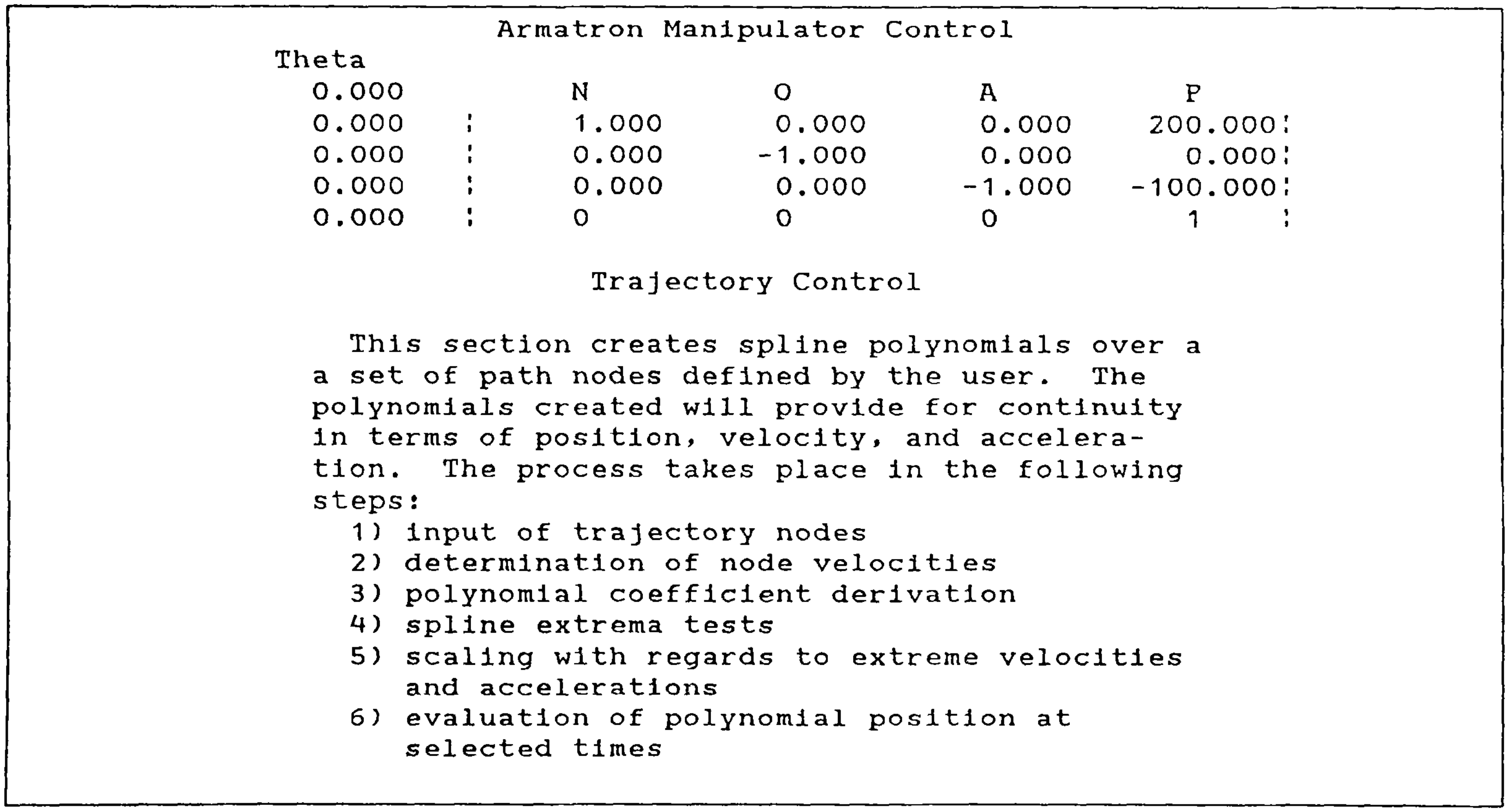
Figure 7.13. Scale and Real Time Intervals

D. THE CONTROLLING PROCEDURE

Procedure `trajectory_control` begins by displaying the introductory screen of Figure 7.14 The array elements allocated for nodes are then set to zero, the value to be used as a default. The iteration of the procedure then directs the execution of the trajectory determination and evaluation steps outlined in this chapter. This process is repeated until the procedure is instructed to stop.

```
dsply_trajectory_introduction ( );
wait_then_erase (9);
for (i = 1; i <= 5; i++)
   for (j = 1; j <= 10; j++)
      theta[i][j] = 0;
do
   {
   n = nodes_and_distances (theta, t);
   f = 1000000;
   for (i = 1; i <= 5; i++)
      {
      lcputs (9, 29, "Trajectory for Joint ");
      cprintf ("%d", i);
      calc_polynomials (n, theta[i], t, a[i], b[i], c[i], d[i],
                        e[i]);
      theta_lb[i] = -360;
      theta_hb[i] =  360;
      critical_positions (n, a[i], b[i], c[i], d[i], e[i], t,
                          theta_lb[i], theta_hb[i]);
      theta_vmax[i] = 100;
      theta_amax[i] = 100;
      f_current = traj_scaling (n, b[i], c[i], d[i], e[i], t,
                                theta_vmax[i], theta_amax[i]);
      if ( f_current < f )
         f = f_current;
      }
   determine_positions (a, b, c, d, e, n, t, f);

   prompt_msg1 = "Continue with a new trajectory determination? (y/n)";
   prompt_msg2 = "";
   qc = prompt_input_char (prompt_msg1, prompt_msg2);
   }
   while ( qc == 'Y' );
wait_then_erase (8);
```

Note that the determination of the scaling factor is performed at this level, as it is to be the minimum of the scaling factors for the trajectories of each joint. The documented listing for the procedures associated with the trajectory control portion of the overall program may be found in Appendix G.



```
                    Armatron Manipulator Control
       Theta
        0.000           N          O          A          P
        0.000   :     1.000      0.000      0.000    200.000:
        0.000   :     0.000     -1.000      0.000      0.000:
        0.000   :     0.000      0.000     -1.000   -100.000:
        0.000   :     0                0          0          1    :

                        Trajectory Control

         This section creates spline polynomials over a
        a set of path nodes defined by the user.  The
        polynomials created will provide for continuity
        in terms of position, velocity, and accelera-
        tion.  The process takes place in the following
        steps:
          1) input of trajectory nodes
          2) determination of node velocities
          3) polynomial coefficient derivation
          4) spline extrema tests
          5) scaling with regards to extreme velocities
             and accelerations
          6) evaluation of polynomial position at
             selected times
```

Figure 7.14. Trajectory Control Introductory Display

VIII. CONCLUSIONS

Several shortcomings and areas of further endeavor have been touched on during the course of this work. This chapter reiterates and expands on each of them.

The electronic control circuit used leads to several deficiencies from a control standpoint. For example, the circuit provides a different amount of current to a motor in each direction, as explained in Chapter 2; thus the joints move faster in one direction than the other. The results obtained would be more appealing if this were not the case; some re-design of the circuitry involved would be called for here. The positioning problem solution could be further enhanced by incorporating some sort of feedback in the robot arm, thus eliminating the need for timing and scaling of iterations to degrees. The incorporation of stepper-motors is another possibility here, as position could be maintained with them as well.

Coordinated motion cannot be accomplished with the given configuration, as only a single motor can be controlled at a time due to the use of a decoder. If sufficient lines were made available from the computer, coordinated motion would become possible. The circuit also lacks the ability to drive the motors at different speeds; this problem would be more difficult to overcome. The lack of both coordinated motion and velocity control prevents the velocity and trajectory control techniques developed in Chapters 6 and 7, respectively, from being implemented.

The geometric interpretation for the effect of the transformation matrices in Chapter 4 and for the inadequacies of the manipulator wrist in Chapter 5 could be enhanced by presenting the geometry graphically. Color and motion could be utilized to further enhance what actually transpires with respect to coordinate frames when robotic joints are actuated. An interactive graphics package could be extended to include the depiction of the manipulator as it follows prescribed velocity or trajectory conditions as well. This could serve to compensate in part for the inadequacies of the actual Armatron manipulator configuration.

There are other topics in robotics which have not been covered here. Dynamics is perhaps the area most commonly entered next. This area deals with the physical characteristics and behavior of robot manipulators, such as mass, inertia, etc.; see [Holl80] or [Vuko82] for typical presentations of this material. The background necessary for an understanding of this topic is common to disciplines such as mechanical engineering, but is typically lacking in a computer science background. There is a need here for the presentation of the dynamics aspects of robotics with background material sufficient for the understanding of the concepts involved.

APPENDIX A

HEADER LISTING

```
#define a2     100
#define a3     100
#define d5     100
#define conv   (3.14159 / 180)
#define del    0x08
#define null   0x00
#define cr     0x0D
#define lf     0x0A
#define eof    0x1A
#define hyphen 0x2D
#define space 0x20
#define tolerance 0.01
#define small_tolerance 1E-50
#define pi     3.14159
#define short_pause 400


typedef int     i_1      [ 2];
typedef int     i_2      [ 3];
typedef int     i_3      [ 4];
typedef int     i_4      [ 5];
typedef int     i_5      [ 6];
typedef int     i_6      [ 7];
typedef int     i_7      [ 8];
typedef int     i_8      [ 9];
typedef float   f_1      [ 2];
typedef float   f_2      [ 3];
typedef float   f_3      [ 4];
typedef float   f_4      [ 5];
typedef float   f_5      [ 6];
typedef float   f_6      [ 7];
typedef float   f_12     [13];
typedef float   f_9      [10];
typedef float   f_10     [11];
typedef float   f_3_2    [ 4][ 3];
typedef float   f_4_3    [ 5][ 4];
typedef float   f_4_5    [ 5][ 6];
typedef float   f_5_5    [ 6][ 6];
typedef float   f_6_5    [ 7][ 6];
typedef float   f_5_9    [ 6][10];
typedef float   f_5_10   [ 6][11];
typedef float   f_9_2    [10][ 3];
typedef char    c_1      [ 2];
typedef char    c_6_9    [ 7][10];
typedef char    c_12_18  [13][19];
typedef char    c_3_2_2  [ 4][ 3][ 3];


void  dsply_main_introduction ( );
void  dsply_thetas_noap (int *row, i_4 cols);
void  noap_matrix (f_5 theta, f_3_2 noap, int row, i_4 cols);
void  dsply_main_selection ( );
int   get_option (int hb);


float magnitude (f_2 vector);
int   round (float value);
int   sign (float value);
float square (float value);
float cube_root (float value);
void  wait_then_continue ( );
void  mcputs (int left, int right, char *line);
int   prompt_input_digit (char *prompt_msg);
char  prompt_input_char (char *prompt_msg1, char *prompt_msg2);
float prompt_input_fixed (char *prompt_msg, int i);
float indec (int r, int c);
int   inint (int r, int c);
```

```
void   lputch (int row, int col, char ch);
void   lcprintf (int row, int col, float value);
void   lcprintf8 (int row, int col, float value);
void   lcputs (int row, int col, char *string);
void   erase_prompt (int row);
void   wait_then_erase (int row);
void   save_screen ( );
void   pause  (int thousandths);
void   locate (int row, int col);



void   manual_control (f_5 theta, f_3_2 noap,
                       int noap_row, i_4 noap_cols);
void   dsply_manual_introduction ( );
void   dsply_keyboard (int *row, int *col);
void   monitor_keyboard (f_5 theta, f_3_2 noap, int noap_row,
                         i_4 noap_cols, int row, int col);
void   init_transistor_messages (c_12_18 msgs);
void   move_manual (int transistor, char *msg, char original_key,
                    int row, int col, f_5 theta, int joint);
float select_scale_manual_move (int transistor);



void   joint_variable_control (f_5 theta, f_3_2 noap,
                               int noap_row, i_4 noap_cols);
void   dsply_joint_variable_introduction ( );
void   constraints (char ignore, f_5 theta_min, f_5 theta_max);
void   dsply_joint_variables (i_5 jv_rows, int *jv_col);
void   process_requests (f_5 theta, f_3_2 noap,
                         f_5 theta_min, f_5 theta_max,
                         i_5 jv_rows, int jv_col,
                         int noap_row, i_4 noap_cols);
int    get_joint ( );
float get_angle (float minimum, float maximum);
void   perform_move (int joint, f_5 theta, float desired_position,
                     int row, int col);
int    select_transistor (int joint, float move);
float select_scale (int transistor);



void   position_orientation_control (f_5 theta, f_3_2 noap,
                                     int noap_row, i_4 noap_cols);
void   dsply_position_orientation_introduction ( );
void   dsply_pos_orient_solution (int *arm_row, i_2 arm_cols,
                                  int *theta_row, i_4 theta_cols);
void   get_noap (f_3_2 noap, int row, i_4 cols);
void   sin_cos (f_5 theta, f_5 s, f_5 c);
void   init_names (c_3_2_2 names);
void   get_orientation_vector (int i, c_3_2_2 names, f_3_2 noap,
                               int row, i_4 cols);
void   get_position_vector (c_3_2_2 names, f_3_2 noap, int row, i_4 cols);
void   prompt_input_noap (char *name, float *value);

int    calc_arm_end (f_3_2 noap, f_2 pa, int arm_row, i_2 arm_cols);
void   calc_theta_123_triples (f_2 pa, f_4_3 theta, int row, i_4 cols);
void   calc_theta_3 (f_2 pa, f_4_3 theta, int row, i_4 cols);
void   calc_beta_minus_theta_1 (f_2 pa, f_4_3 theta, f_4 bmt1);
float calc_beta (f_2 pa);
void   calc_theta_1 (float beta, f_4 bmt1, f_4_3 theta,
                     int row, i_4 cols);
void   calc_theta_2 (f_2 pa, f_4_3 theta, int row, i_4 cols);
int    calc_theta_45_pairs (f_3_2 noap, f_4_3 theta, f_4_5 accepted_theta,
                            int row, i_4 cols);
float calc_theta_4 (f_2 a, f_3 s, f_3 c);
float calc_theta_5 (f_2 n, f_2 o, f_3 s, f_3 c);

char   prompt_for_move (int accepted, f_4_5 accepted_theta,
                        f_5 move_theta);
void   dsply_prompt_for_move (int *row, i_8 cols);
float  min_constraint (int joint);
float  max_constraint (int joint);
char   one_solution (int index, f_4_5 accepted_theta, f_5 move_theta);
char   multiple_solutions (int accepted, f_4_5 accepted_theta,
                           int inbounds, i_4 inbounds_index, i_4 out,
                           f_5 move_theta);
void   position_orientation_move (f_5 theta, f_5 move_theta);
void   dsply_pos_orient_move (int *row, i_2 cols);



void   velocity_control (f_5 theta, f_3_2 noap, int row, i_4 cols);
void   dsply_velocity_introduction ( ) ;
void   dsply_vc_selection ( );
void   get_theta (f_5 theta, int row, int cols);

void   for_sol_via_jac (f_5 theta, f_5 dtheta);
void   dsply_jacobian (int *row, i_6 cols);
void   calc_jacobian (f_5 s, f_5 c, f_6_5 jacobian, int row, i_5 cols);
void   get_delta_theta (f_5 dtheta, int r, int c);
void   calc_list_rates (f_5 dtheta, f_6 drate, f_6_5 jacobian,
                        int r, int c);
char   cont (char *msg);

void   rev_sol_via_ij  (f_5 theta, f_6 delta_trans_rot);
void   dsply_rsvij_jacobian (int *row, i_5 cols);
int    get_required_rates (f_6 delta_trans_rot, f_6_5 jacobian, i_6 used,
                           f_6_5 jacobian_reduced, f_6 delta_tr_reduced,
                           int row, i_5 cols);
int    over_determined_case (f_6_5 jacobian, f_6 delta_trans_rot,
                             f_5 delta_theta);
int    under_determined_case (int total, f_6_5 jacobian_reduced,
                              f_6 delta_tr_reduced, f_5 delta_theta);
void   matrix_by_matrix (int total, f_6_5 mparm, f_5_5 m, char *subhead);
void   dsply_m (char *subhead, int *row, i_5 cols);
void   matrix_by_vector (f_6_5 jac_parm, int total, f_5 vec, f_5 result,
                         char *subhead);
void   dsply_m_by_v (char *subhead, int *row, i_7 cols);
void   list_input_output (f_6 delta_tran_rot, i_6 used, f_5 delta_theta);
void   dsply_in_out (i_1 row, int *col);
int    solve_simul_eqns_myv (f_5_5 m, f_5 v, int n, f_5 y, char *subhead)
void   dsply_soln_myv (char *subhead, int *row, i_7 cols);
int    interchange_rows (f_5_5 m, f_5 v, int k, int n,
                         int row, i_7 cols);
void   zero_column_k (f_5_5 m, f_5 v, int k, int n, int row, i_7 cols);
void   solve_y_vector (f_5_5 m, f_5 v, int n, f_5 y, int row, i_7 cols);

void   rev_sol_via_deriv (f_5 theta, f_3_2 noap, f_6 delta_trans_rot);
void   dsply_rsvd (int *mr, i_3 mc, i_1 vr, i_1 vc);
void   get_delta_trans_rot (f_6 delta_trans_rot, int vr, int vc);
void   calc_delta_noap (f_3_2 noap, f_6 delta_trans_rot, f_3_2 dnoap,
       i_3 mc, int mr);
void   calc_delta_theta (f_5 s, f_5 c, f_3_2 dnoap, f_3_2 noap,
                         f_5 dtheta, int vr, int vc);
void   delta_theta312 (f_5 s, f_5 c, f_3_2 dnoap, f_3_2 noap, f_5 dtheta,
                       int *div_zero);
void   delta_theta4 (f_5 s, f_5 c, f_3_2 dnoap, f_3_2 noap, f_5 dtheta);
void   delta_theta5 (f_5 s, f_5 c, f_3_2 dnoap, f_3_2 noap, f_5 dtheta);



void   trajectory_control ( );
void   dsply_trajectory_introduction ( );
int    nodes_and_distances (f_5_10 theta, f_9 t);
void   dsply_nodes_dists (int *row, i_6 cols);
int    input_nodes (f_5_10 p, int row, i_6 cols);
void   calc_distance (int n, f_5_10 p, f_9 t, int row, int col);
void  calc_polynomials (int n, f_10 p, f_9 t,
                        f_9 a, f_9 b, f_9 c, f_9 d, f_9 e);

void  calc_node_velocities (int n, f_10 p, f_9 t, f_9 vel);
void  dsply_node_velocities (int *row, i_4 cols);
void  equate_quartic_cubic_accs (f_9 t, f_10 p, f_9_2 coeff, f_9 rhs,
                                 int row, i_4 cols);
void  equate_cubic_accs (int n, f_9 t, f_10 p, f_9_2 coeff, f_9 rhs,
                         int row, i_4 cols);
void  equate_cubic_quartic_accs (int n, f_9 t, f_10 p, f_9_2 coeff,
                                 f_9 rhs, int row, i_4 cols);
void  forward_eliminate_term1 (int n, f_9_2 coeff, f_9 rhs,
                               int row, i_4 cols);
void  backward_eliminate_term3 (int n, f_9_2 coeff, f_9 rhs, f_9 vel,
                                int row, i_4 cols);

void  calc_coefficients (int n, f_10 p, f_9 t, f_9 vel,
                         f_9 a, f_9 b, f_9 c, f_9 d, f_9 e);
void  dsply_coefficients (int *row, i_5 cols);
void  calc_starting_quartic (f_10 p, f_9 t, f_9 vel,
                             f_9 a, f_9 b, f_9 c, f_9 d, f_9 e,
                             int row, i_5 cols);
void  calc_intermediate_cubics (int n, f_10 p, f_9 t, f_9 vel,
                                f_9 a, f_9 b, f_9 c, f_9 d, f_9 e,
                                int row, i_5 cols);
void  calc_ending_quartic (int n, f_10 p, f_9 t, f_9 vel,
                           f_9 a, f_9 b, f_9 c, f_9 d, f_9 e,
                           int row, i_5 cols);

void  critical_positions (int n, f_9 a, f_9 b, f_9 c, f_9 d, f_9 e,
                          f_9 t, int lb, int hb);
void  dsply_crit_pos (int *row, i_7 cols);
void  terminal_crit_pos (int i, float a, float b, float c, float d,
                         float e, float t, float lb, float hb,
                         int row, i_7 cols);
void  intermediate_crit_pos (int i, float a, float b, float c, float d,
                             float t, float lb, float hb,
                             int row, i_7 cols);
float eval_cp (float a, float b, float c, float d, float e,
               float tcp, float t, int row, int tcol, int pcol);
void  check_range (float lb, float hb, float cp, int row, int col);

float traj_scaling (int n, f_9 b, f_9 c, f_9 d, f_9 e, f_9 t,
                    float theta_maxv, float theta_maxa);
void  dsply_traj_scaling (int *row, i_7 cols);
float terminal_crit_vel (int i, float b, float c, float d, float e,
                         float t, int row, i_6 cols);
float intermediate_crit_vel (int i, float b, float c, float d, float t,
                             int row, i_6 cols);
float eval_cv (float b, float c, float d, float e, float tcv, float t,
               int row, int tcol, int vcol);
float terminal_crit_acc (float c, float d, float e, float t,
                         int row, i_6 cols);
float intermediate_crit_acc (float c, int row, i_6 cols);

void  determine_positions (f_5_9 a, f_5_9 b, f_5_9 c, f_5_9 d, f_5_9 e,
                           int n, f_9 t, float f);
void  dsply_det_positions (i_2 row, i_4 time_cols, i_6 eval_cols,
                           int *scale_col);
float get_time ( );


#include  <stdio.h>
#include    <dos.h>
#include   <time.h>
#include   <math.h>
#include <string.h>
#include  <ctype.h>
```

APPENDIX B

MAIN PROCEDURE LISTING

```
#include <c:\ed's\header.c>

FILE  *fptr;
char  qsave_screen;

void main ( )
   {
   int    row;
   i_4    cols;
   int    i;
   f_5    theta;
   f_5    s;
   f_5    c;
   f_3_2  noap;
   int    opt;

   outportb (888, 0);
   fptr = fopen ("SCREENS.OUT", "w+t");
   dsply_main_introduction ( );
   locate (23, 55);
   qsave_screen = toupper (getch ( ));
   lputch (23, 55, qsave_screen);

   wait_then_erase (1);
   dsply_thetas_noap (&row, cols);
   for (i = 1; i <= 5; i++)
      theta[i] = 0;
   noap_matrix (theta, noap, row, cols);
   dsply_main_selection ( );
   while ( (opt = get_option(5)) != 0 )
      {
      wait_then_erase (8);
      switch (opt)
         {
         case 1 : manual_control (theta, noap, row, cols);
                  break;
         case 2 : joint_variable_control (theta, noap, row, cols);
                  break;
         case 3 : position_orientation_control (theta, noap, row, cols);
                  break;
         case 4 : velocity_control (theta, noap, row, cols);
                  break;
         case 5 : trajectory_control ( );
                  break;
         }
      dsply_main_selection ( );
      }
   fputc (eof, fptr);
   fclose (fptr);
   }

void dsply_main_introduction ( )
   {
   int  lm;
   int  rm;

   lm = 16;
   rm = 16;
   locate (0, 0);
   mcputs (lm, rm, "          Armatron Manipulator Control          ");
   mcputs (lm, rm, "                   Version 1.1                  ");
   mcputs (lm, rm, "          Edward Hammerand  March 1990          ");
   mcputs (lm, rm, "                                                ");
   mcputs (lm, rm, "  This program provides for the control of the  ");
   mcputs (lm, rm, "Armatron manipulator in one of three manners:   ");
   mcputs (lm, rm, "  1) The manipulator may be controlled directly ");
   mcputs (lm, rm, "      using keyboard input                      ");
   mcputs (lm, rm, "  2) The settings of the joint variables may be ");
   mcputs (lm, rm, "      input directly                            ");
   mcputs (lm, rm, "  3) A desired position and orientation of the  ");
   mcputs (lm, rm, "     manipulator may be specified; from this, a ");
   mcputs (lm, rm, "     solution set will be derived and a move    ");
   mcputs (lm, rm, "     attempted if possible and desired          ");
   mcputs (lm, rm, "Additionally, support is given for the calcula- ");
   mcputs (lm, rm, "tion of manipulator velocities and trajectories,");
   mcputs (lm, rm, "although these are not directly implemented for ");
   mcputs (lm, rm, "the robot arm.                                  ");
   mcputs (lm, rm, "  If the manipulator arm is not aligned to its  ");
   mcputs (lm, rm, "home orientation, use the manual switches to    ");
   mcputs (lm, rm, "align it at this time.                          ");
   mcputs (lm, rm, "  If desired, the screens displayed during the  ");
   mcputs (lm, rm, "course of program execution may be saved to the ");
   mcputs (lm, rm, "file SCREENS.OUT; save screens (y/n)?           ");
   mcputs (lm, rm, "                                               ");
   }

void dsply_thetas_noap (int *row, i_4 cols)
   {
   locate (1, 0);
   cputs ("              Theta                            ");
   cputs ("                                  ");
   cputs ("               T1               N          O    ");
   cputs ("      A          P                ");
   cputs ("               T2         |                     ");
   cputs ("                       |          ");
   cputs ("               T3         |                     ");
   cputs ("                       |          ");
   cputs ("               T4         |                     ");
   cputs ("                       |          ");
   cputs ("               T5         |     0          0    ");
   cputs ("      0          1     |          ");

   cols[0] = 25;
   cols[1] = 36;
   cols[2] = 47;
   cols[3] = 58;
   cols[4] = 11;
   *row    =  3;
   }

void noap_matrix (f_5 theta, f_3_2 noap, int row, i_4 cols)
   {
   int  i;
   int  j;
   f_5  s;
   f_5  c;

   for (i = 1; i <= 5; i++)
      {
      lcprintf (row+i-2, cols[4], theta[i]);
      }

   sin_cos (theta, s, c);

   noap[0][0] = ((c[1]*c[2]*c[3] - s[1]*s[3])*c[4]
                 - c[1]*s[2]*s[4])*c[5]
              + (c[1]*c[2]*s[3] + s[1]*c[3])*s[5];
   noap[0][1] = ((s[1]*c[2]*c[3] + c[1]*s[3])*c[4]
                 - s[1]*s[2]*s[4])*c[5]
              + (s[1]*c[2]*s[3] - c[1]*c[3])*s[5];
   noap[0][2] = (s[2]*c[3]*c[4] + c[2]*s[4])*c[5] + s[2]*s[3]*s[5];

   noap[1][0] = -((c[1]*c[2]*c[3] - s[1]*s[3])*c[4]
                 - c[1]*s[2]*s[4])*s[5]
              + (c[1]*c[2]*s[3] + s[1]*c[3])*c[5];
   noap[1][1] = -((s[1]*c[2]*c[3] + c[1]*s[3])*c[4]
                 - s[1]*s[2]*s[4])*s[5]
              + (s[1]*c[2]*s[3] - c[1]*c[3])*c[5];
   noap[1][2] = -(s[2]*c[3]*c[4] + c[2]*s[4])*s[5] + s[2]*s[3]*c[5];
```

```
    noap[2][0] = (c[1]*c[2]*c[3] - s[1]*s[3])*s[4] + c[1]*s[2]*c[4];
    noap[2][1] = (s[1]*c[2]*c[3] + c[1]*s[3])*s[4] + s[1]*s[2]*c[4];
    noap[2][2] =  s[2]*c[3]*s[4] - c[2]*c[4];

    noap[3][0] = d5*((c[1]*c[2]*c[3] - s[1]*s[3])*s[4] + c[1]*s[2]*c[4])
               + a3*(c[1]*c[2]*c[3] - s[1]*s[3]) + a2*c[1]*c[2];
    noap[3][1] = d5*((s[1]*c[2]*c[3] + c[1]*s[3])*s[4] + s[1]*s[2]*c[4])
               + a3*(s[1]*c[2]*c[3] + c[1]*s[3]) + a2*s[1]*c[2];
    noap[3][2] = d5*(s[2]*c[3]*s[4] - c[2]*c[4]) + (a3*c[3] + a2)*s[2];

    for (i = 0; i <= 3; i++)
       for (j = 0; j <= 2; j++)
          lcprintf (row+j, cols[i], noap[i][j]);
    }

void sin_cos (f_5 theta, f_5 s, f_5 c)
    {
    int  i;

    for (i = 1; i <= 5; i++)
       {
       s[i] = sin (theta[i] * conv);
       c[i] = cos (theta[i] * conv);
       }
    }

void dsply_main_selection ( )
    {
    int  row;
    int  col;

    row =  8;
    col = 22;
    lcputs (row,    col, "Armatron Manipulator Control Options");
    lcputs (row+ 2, col, "1:  Manual Control");
    lcputs (row+ 4, col, "2:  Joint Variable Control");
    lcputs (row+ 6, col, "3:  Position-Orientation Control");
    lcputs (row+ 8, col, "4:  Velocity Control");
    lcputs (row+10, col, "5:  Trajectory Control");
    lcputs (row+12, col, "0:  Terminate Manipulator Control");
    }

int get_option (int hb)
    {
    int  opt;

    do
       opt = prompt_input_digit ("Select Option:");
       while ( opt > hb );
    locate (22, 22);
    cprintf ("Option %d has been selected", opt);
    return (opt);
    }



float magnitude (f_2 vector)
    {
    int    i;
    float  sum;

    sum = 0;
    for (i = 0; i <= 2; i++)
       sum += square (vector[i]);
    return (sqrt(sum));
    }

int round (float value)
    {
    value = value + sign (value) * 0.5;
    return ( (int) value );
    }

int sign (float value)
    {
    int  value_sign;

    if ( value > 0 )
            value_sign = 1;
        else
            if ( value < 0 )
                    value_sign = -1;
                else
                    value_sign =  0;
    return (value_sign);
    }

float square (float value)
    {
    return (pow(value, 2));
    }

float cube_root (float value)
    {
    float  result;

    if (value > 0)
            result = pow (value, 1/3);
        else
            if (value < 0)
                    result = -pow (-value, 1/3);
                else
                    result = 0;
    return (result);
    }

void wait_then_continue ( )
    {
    save_screen ( );
    lcputs (24, 28, "Press any key to proceed");
    locate (24, 54);
    getch ( );
    erase_prompt (24);
    }

void mcputs (int left, int right, char *line)
    {
    int  i;

    for (i = 1; i <= left; i++)
        cputs (" ");
    cputs (line);
    for (i = 1; i <= right; i++)
        cputs (" ");
    }

int prompt_input_digit (char *prompt_msg)
    {
    char  ch;
    c_1   ch_string;

    do
        {
        lcputs (23, 20, prompt_msg);
        locate (23, strlen(prompt_msg)+22);
        ch = getch( );
        }
        while ( (ch < '0') ¦ (ch > '9') );
    erase_prompt (23);
    ch_string[0] = ch;
```

```
   ch_string[1] = '';
   return (atoi(ch_string));
   }

char prompt_input_char (char *prompt_msg1, char *prompt_msg2)
   {
   int   r;
   int   c;
   char  ch;

   lcputs (23, 20, prompt_msg1);
   lcputs (24, 20, prompt_msg2);
   locate (23, strlen(prompt_msg1)+21);
   ch = toupper ( getch ( ) );
   erase_prompt (23);
   return (ch);
   }

float prompt_input_fixed (char *prompt_msg, int i)
   {
   float  value;

   lcputs (23, 20, prompt_msg);
   cprintf (" %d <Snnn.nnn>:  ", i);
   lcputs (24, 20, "(<Return> only to leave value unchanged)");
   value = indec (23, 25+strlen(prompt_msg));
   erase_prompt (23);
   return (value);
   }

float indec (int r, int c)
   {
   char   instring [8];
   int    len;
   int    currcol;
   char   ch;
   int    ch_ok;
   float  value;
   int    decpt;
   int    fraclen;
   int    done;

   locate (r, c);
   instring[0] = null;
   len         = 0;
   currcol     = c;
   decpt       = 0;
   fraclen     = 0;
   done        = 0;

   while (!done)
      {
      ch = getch ();
      ch_ok = 0;
      value = atof (instring);
      if (((ch >= '0') & (ch <= '9')) & ((value < 100) ¦ (decpt == 1)) &
          (fraclen < 3))
            {
            if (decpt == 1)
               fraclen++;
            ch_ok = 1;
            }
         else
            if ((ch == '.') & (decpt == 0))
                  {
                  decpt = 1;
                  ch_ok = 1;
                  }
               else
                  if ( ((ch == '+') ¦ (ch == '-')) & (len == 0) )
```

```
                     ch_ok = 1;
      if (ch_ok == 1)
            {
            instring[len] = ch;
            len++;
            instring[len] = null;
            lputch (r, currcol, ch);
            currcol++;
            }
         else
            if ((ch == del) & (len > 0))
                  {
                  len--;
                  if (instring[len] == '.')
                     decpt = 0;
                  if (fraclen > 0)
                     fraclen--;
                  instring[len] = null;
                  lputch (r, currcol, space);
                  currcol--;
                  lputch (r, currcol, space);
                  locate (r, currcol);
                  }
               else
                  if (ch == cr)
                     done = 1;
      }
   if (len == 0)
         value = 1000;
      else
         value = atof (instring);
   return (value);
   }

int inint (int row, int col)
   {
   char  instring [4];
   int   len;
   int   currcol;
   char  ch;
   int   ch_ok;
   int   value;
   int   done;

   locate (row, col);
   instring[0] = null;
   len          = 0;
   currcol      = col;
   done         = 0;

   while (!done)
      {
      ch = getch ();
      ch_ok = 0;
      value = atoi (instring);
      if ( (ch >= '0') & (ch <= '9') )
            ch_ok = 1;
         else
            if ( ((ch == '+') ¦ (ch == '-')) & (len == 0) )
               ch_ok = 1;
      if ( ch_ok == 1 )
            {
            instring[len] = ch;
            len++;
            instring[len] = null;
            lputch (row, currcol, ch);
            currcol++;
            }
         else
            if ( (ch == del) & (len > 0) )
```

```
                  {
                  len--;
                  instring[len] = null;
                  lputch (row, currcol, space);
                  currcol--;
                  lputch (row, currcol, space);
                  locate (row, currcol);
                  }
               else
                  if (ch == cr)
                     done = 1;
         }
      if ( len == 0 )
            value = 0;
         else
            value = atoi (instring);
      return (value);
      }

   void lputch (int   row,
                int   col,
                char  ch)
      {
      locate (row, col);
      putch (ch);
      }

   void lcprintf (int row, int col, float value)
      {
      locate (row, col);
      if ( ( fabs(value) >= 0.01  ) ¦ ( value == 0 ) )
            cprintf ("%10.3f", value);
         else
            cprintf ("%10.4E", value);
      }

   void lcprintf8 (int row, int col, float value)
      {
      locate (row, col);
      if ( ( fabs(value) >= 0.01  ) ¦ ( value == 0 ) )
            cprintf ("%8.3f", value);
         else
            cprintf ("%8.1E", value);
      }

   void lcputs (int row, int col, char *string)
      {
      locate (row, col);
      cputs (string);
      }

   void erase_prompt (int row)
      {
      int  i;

      for (i = row; i <= 23; i++)
         {
         locate (i, 0);
         cprintf ("%80c", ' ');
         }
      locate (24, 0);
      cprintf ("%79c", ' ');
      }

   void wait_then_erase (int row)
      {
      int  r;
      int  c;

      save_screen ( );
      lcputs (24, 28, "Press any key to proceed");
      locate (24, 54);
      getch ();
      c = 0;
      for (r = row; r <= 23; r++)
         {
         locate (r, c);
         cprintf ("%80c", ' ');
         }
      locate (24, 0);
      cprintf ("%79c", ' ');
      }

   void save_screen ( )
      {
      int       row;
      int       col;
      unsigned  offset;
      char      ch;

      if ( qsave_screen == 'Y' )
         {
         for (row = 0; row <= 24; row++)
            {
            for (col = 0; col <= 79; col++)
               {
               offset = row*160 + col*2;
               ch = peekb(47104, offset);
               fputc (ch, fptr);
               }
            fputc (lf, fptr);
            }
         for (col = 0; col <= 79; col++)
            fputc (hyphen, fptr);
         fputc (lf, fptr);
         }
      }

   void pause (int thousandths)
                                       /* PAUSE : delay for the input   */
                                       /*         number of 1/1000's of */
                                       /*         seconds               */
      {
      unsigned long  ticks;
      unsigned long  target;

      union REGS     i, o;

      i.h.ah = 0;
      int86 (26, &i, &o);
      ticks = (o.x.cx << 16) ¦ o.x.dx;
      target = ticks + (thousandths / 55);

      while (ticks < target)
         {
         i.h.ah = 0;
         int86 (26, &i, &o);
         ticks = (o.x.cx << 16) ¦ o.x.dx;
         }
      }

   void locate (int row, int col)
                                       /* LOCATE : move cursor to (r,c)  */
                                       /*          r = 0 to 24           */
                                       /*          c = 0 to 79           */
      {
      union REGS  i;

      i.h.ah = 2;
      i.h.bh = 0;
```

```
i.h.dh = row;
i.h.dl = col;
int86 (16, &i, &i);
}
```

APPENDIX C

MANUAL CONTROL PROCEDURES LISTING

```
#include <c:\ed's\header.c>

void manual_control (f_5 theta, f_3_2 noap, int noap_row, i_4 noap_cols)
   {
   int  row;
   int  col;

   dsply_manual_introduction ( );
   wait_then_erase (9);
   dsply_keyboard (&row, &col);
   monitor_keyboard (theta, noap, noap_row, noap_cols, row, col);
   erase_prompt (23);
   locate (23, 0);
   mcputs (28, 27, "Manual Control Terminated");
   wait_then_erase (8);
   }

void dsply_manual_introduction ( )
   {
   int  lm;
   int  rm;

   locate (8, 0);
   lm = 16;
   rm = 16;
   mcputs (lm, rm, "                 Manual Control                 ");
   mcputs (lm, rm, "                                                ");
   mcputs (lm, rm, "  The movement of each of the five joints and   ");
   mcputs (lm, rm, "the gripper of the Armatron manipulator is      ");
   mcputs (lm, rm, "controlled from the keyboard by a pair of keys. ");
   mcputs (lm, rm, "                                                ");
   mcputs (lm, rm, "  To effect movement of a joint, press and hold ");
   mcputs (lm, rm, "down one of the keys controlling the joint; the ");
   mcputs (lm, rm, "release of the key terminates movement.  The    ");
   mcputs (lm, rm, "amount of time the key is held down is monitored");
   mcputs (lm, rm, "and used to update the joint variable involved. ");
   mcputs (lm, rm, "                                                ");
   mcputs (lm, rm, "Note:  At times, a motor may stall; should this ");
   mcputs (lm, rm, "       occur, immediately release the key to    ");
   mcputs (lm, rm, "       avoid ruining a transistor.              ");
   }

void dsply_keyboard (int *row, int *col)
   {
   int  lm;
   int  rm;

   locate (9, 0);
   lm = 16;
   rm = 16;
   mcputs (lm, rm, "                                                ");
   mcputs (lm, rm, "Gripper Open                       Gripper Close");
   mcputs (lm, rm, "|                                              |");
   mcputs (lm, rm, "|       E*-Wrist Up   (4)  Wrist Down-*I       |");
   mcputs (lm, rm, "|                                              |");
   mcputs (lm, rm, "A*  S*  D*  F*  G*-Arm(2) Arm-*H  *J  *K  *L  *;");
   mcputs (lm, rm, "    |   |   |      Up   Down       |   |   |    ");
   mcputs (lm, rm, "    |   |   |                      |   |   |    ");
   mcputs (lm, rm, "    |   |   Arm Left  (1)  Arm Right   |   |    ");
   mcputs (lm, rm, "    |   |                              |   |    ");
   mcputs (lm, rm, "    |   Elbow Left    (3)   Elbow Right    |    ");
   mcputs (lm, rm, "    |                                      |    ");
   mcputs (lm, rm, "    Wrist Rotate Left (5) Wrist Rotate Right    ");
   mcputs (lm, rm, "                                                ");
   mcputs (lm, rm, "Select:                                         ");
   mcputs (lm, rm, "  <Press the space bar to terminate control>    ");

   *row = 23;
   *col = 25;
   }
```

```
void monitor_keyboard (f_5 theta, f_3_2 noap, int noap_row,
                       i_4 noap_cols, int row, int col)
   {
   char      key;
   c_12_18   msgs;
   int       transistor;
   int       joint;

   init_transistor_messages (msgs);
   do
      {
      locate (row, col);
      while ( !kbhit( ) );
      key = toupper(getch( ));
      switch (key)
         {
                                            /* Arm Down - Up */
         case 'H' : transistor =  7;
                    joint = 2;
                    break;
         case 'G' : transistor =  8;
                    joint = 2;
                    break;
                                            /* Arm Right - Left */
         case 'J' : transistor = 12;
                    joint = 1;
                    break;
         case 'F' : transistor = 11;
                    joint = 1;
                    break;
                                            /* Elbow Left - Right */
         case 'D' : transistor =  6;
                    joint = 3;
                    break;
         case 'K' : transistor =  5;
                    joint = 3;
                    break;
                                            /* Wrist Down - Up */
         case 'I' : transistor =  1;
                    joint = 4;
                    break;
         case 'E' : transistor =  2;
                    joint = 4;
                    break;
                                            /* Wrist Rotate Left - Right */
         case 'S' : transistor =  9;
                    joint = 5;
                    break;
         case 'L' : transistor = 10;
                    joint = 5;
                    break;
                                            /* Gripper Open - Close */
         case 'A' : transistor =  4;
                    joint = 0;
                    break;
         case ';' : transistor =  3;
                    joint = 0;
                    break;

         default  : transistor =  0;
                    joint = 0;
         }
      if ( transistor != 0 )
         move_manual (transistor, msgs[transistor], key, row, col,
                      theta, joint);
      if ( joint > 0 )
         noap_matrix (theta, noap, noap_row, noap_cols);
      }
      while ( key != space );
   }

void init_transistor_messages (c_12_18 msgs)
   {
   strcpy (msgs[ 1], "Wrist Down        ");
   strcpy (msgs[ 2], "Wrist Up          ");
   strcpy (msgs[ 3], "Gripper Close     ");
   strcpy (msgs[ 4], "Gripper Open      ");
   strcpy (msgs[ 5], "Elbow Right       ");
   strcpy (msgs[ 6], "Elbow Left        ");
   strcpy (msgs[ 7], "Arm Down          ");
   strcpy (msgs[ 8], "Arm Up            ");
   strcpy (msgs[ 9], "Wrist Rotate Left ");
   strcpy (msgs[10], "Wrist Rotate Right");
   strcpy (msgs[11], "Arm Left          ");
   strcpy (msgs[12], "Arm Right         ");
   }

void move_manual (int transistor, char *msg, char original_key,
                  int row, int col, f_5 theta, int joint)
   {
   char   key;
   float  degree_scale;
   int    i;

   degree_scale = select_scale_manual_move (transistor);
   i = 0;
   lcputs (row, col, msg);
   locate (row, col);
   pause (500);
   outportb (888, transistor);
   do
      {
      i++;
      pause (110);
      key = null;
      while (kbhit ( ))
         key = toupper (getch ( ));
      }
      while ( key == original_key );
   outportb (888, 0);
   lcputs (row, col, "                  ");
   if ( joint > 0 )
      theta[joint] += (float) i / degree_scale;
   }

float select_scale_manual_move (int transistor)
   {
   float  scale;

   switch (transistor)
      {                                 /* Wrist Down - Up */
      case  1 : scale = -30 /  200.0;
                break;
      case  2 : scale =  36 /  200.0;
                break;
                                        /* Gripper Close - Open */
      case  3 : scale = 1;
                break;
      case  4 : scale = 1;
                break;
                                        /* Elbow Right - Left */
      case  5 : scale = -41 /  180.0;
                break;
      case  6 : scale =  41 /  180.0;
                break;
                                        /* Arm Down - Up */
      case  7 : scale = -29 /   35.0;
                break;
      case  8 : scale =  27 /   35.0;
```

```
            break;
                                              /* Wrist Rotate Left - Right */
   case  9 : scale = -68 / 1080.0;
            break;
   case 10 : scale =  66 / 1080.0;
            break;
                                              /* Arm Left - Right */
   case 11 : scale =  71 /  360.0;
            break;
   case 12 : scale = -65 /  360.0;
            break;
   }
return (scale);
}
```

APPENDIX D

JOINT VARIABLE CONTROL PROCEDURES LISTING

```
#include <c:\ed's\header.c>

void joint_variable_control (f_5 theta, f_3_2 noap,
                             int noap_row, i_4 noap_cols)
   {
   char  ignore;
   f_5   theta_min;
   f_5   theta_max;
   i_5   jv_rows;
   int   jv_col;
   int   i;

   dsply_joint_variable_introduction ( );
   locate (23, 42);
   ignore = toupper(getch( ));
   lputch (23, 42, ignore);
   constraints (ignore, theta_min, theta_max);
   wait_then_erase (9);
   dsply_joint_variables (jv_rows, &jv_col);
   for (i = 1; i <= 5; i++)
      lcprintf (jv_rows[i], jv_col, theta[i]);
   process_requests (theta, noap, theta_min, theta_max,
                     jv_rows, jv_col, noap_row, noap_cols);
   erase_prompt (23);
   locate (23, 0);
   mcputs (24, 23, "Joint-Variable Control Terminated");
   wait_then_erase (8);
   }

void dsply_joint_variable_introduction ( )
   {
   int  lm;
   int  rm;

   locate (8, 0);
   lm = 16;
   rm = 16;
   mcputs (lm, rm, "               Joint-Variable Control              ");
   mcputs (lm, rm, "                                                   ");
   mcputs (lm, rm, "  The movement of each of the five joints of the");
   mcputs (lm, rm, "Armatron manipulator is controlled by specifying");
   mcputs (lm, rm, "a joint and angle via the keyboard.              ");
   mcputs (lm, rm, "                                                   ");
   mcputs (lm, rm, "  Use the manual switches to align the robot arm");
   mcputs (lm, rm, "now, if necessary.                                ");
   mcputs (lm, rm, "                                                   ");
   mcputs (lm, rm, "Note:  At times a motor may stall; should this   ");
   mcputs (lm, rm, "       occur, immediately press the space bar to");
   mcputs (lm, rm, "       avoid ruining a transistor.                ");
   mcputs (lm, rm, "                                                   ");
   mcputs (lm, rm, "  The constraints placed on the joint variables ");
   mcputs (lm, rm, "may be ignored for computation purposes.  Ignore");
   mcputs (lm, rm, "joint constraints? (y/n)                          ");
   }

void constraints (char ignore, f_5 theta_min, f_5 theta_max)
   {
   int  i;

   if ( ignore != 'Y' )
         {
         theta_min[1] = -360;
         theta_max[1] =  360;
         theta_min[2] =   -5;
         theta_max[2] =   30;
         theta_min[3] =  -90;
         theta_max[3] =   90;
         theta_min[4] =  -10;
         theta_max[4] =  190;
         theta_min[5] = -360;
         theta_max[5] =  360;
         }
      else
         for (i = 1; i <= 5; i++)
            {
            theta_min[i] = -360;
            theta_max[i] =  360;
            }
   }

void dsply_joint_variables (i_5 jv_rows, int *jv_col)
   {
   int  lm;
   int  rm;

   lm = 19;
   rm = 19;
   locate (9, 0);
   mcputs (lm, rm, "                                              ");
   mcputs (lm, rm, "       Joint                           Angle ");
   mcputs (lm, rm, "  1:  Arm Right/Left                         ");
   mcputs (lm, rm, "      (-360 to +360)                         ");
   mcputs (lm, rm, "  2:  Arm Down/Up                            ");
   mcputs (lm, rm, "      (  -5 to  +30)                         ");
   mcputs (lm, rm, "  3:  Elbow Right/Left                       ");
   mcputs (lm, rm, "      ( -90 to  +90)                         ");
   mcputs (lm, rm, "  4:  Wrist Down/Up                          ");
   mcputs (lm, rm, "      ( -10 to +190)                         ");
   mcputs (lm, rm, "  5:  Wrist Rotate Left/Right                ");
   mcputs (lm, rm, "      (-360 to +360)                         ");
   mcputs (lm, rm, "  0:  End Joint-Variable Control             ");
   mcputs (lm, rm, "                                              ");

   jv_rows[1] = 11;
   jv_rows[2] = 13;
   jv_rows[3] = 15;
   jv_rows[4] = 17;
   jv_rows[5] = 19;
   *jv_col    = 51;
   }

void process_requests (f_5 theta, f_3_2 noap,
                       f_5 theta_min, f_5 theta_max,
                       i_5 jv_rows, int jv_col,
                       int noap_row, i_4 noap_cols)
   {
   int    joint;
   float  angle;
   float  move_degrees;

   joint = get_joint ( );
   while ( joint != 0 )
      {
      angle = get_angle (theta_min[joint], theta_max[joint]);
      erase_prompt (23);
      locate (23, 20);
      cprintf ("Moving Joint %d to angle %8.3f", joint, angle);
      perform_move (joint, theta, angle, jv_rows[joint], jv_col);
      lcputs (23, 20, "                                  ");

      noap_matrix (theta, noap, noap_row, noap_cols);

      joint = get_joint ( );
      }
   }

int get_joint ( )
   {
   int  joint;
```

```
    do
       joint = prompt_input_digit ("Select Joint:");
       while ( joint > 5 );
    locate (24, 20);
    cprintf ("Joint %d has been selected", joint);
    return (joint);
    }

 float get_angle (float minimum, float maximum)
    {
    float  angle;

    do
       {
       lcputs (23, 20, "Enter angle <Snnn.nnn>:  ");
       angle = (indec (23, 45));
       if ( angle == 1000 )
          angle = 0;
       if ( (angle < minimum) ¦ (angle > maximum) )
          {
          locate (23, 10);
          cprintf ("Angle %8.3f out of range for the joint; ", angle);
          cputs ("check ranges above");
          pause (3000);
          locate (23, 10);
          cprintf ("%61c", ' ');
          }
       }
       while ( (angle < minimum) ¦ (angle > maximum) );
    return (angle);
    }

 void perform_move (int joint, f_5 theta, float desired_position,
                    int row, int col)
    {
    float  move_degrees;
    int    transistor;
    float  degree_scale;
    int    iterations;
    float  degrees_per_iteration;
    int    i;

    move_degrees = desired_position - theta[joint];
    transistor = select_transistor (joint, move_degrees);
    degree_scale = select_scale (transistor);
    iterations = round (fabs(move_degrees) * degree_scale);
    degrees_per_iteration = sign (move_degrees) / degree_scale;
    lcputs (row, col, "    Moving");
    i = 0;
    outportb (888, transistor);
    while ( (!kbhit( )) & (i < iterations) )
       i = i + 1;
    outportb (888, 0);
    if ( i == iterations )
          theta[joint] = desired_position;
       else
          {
          getch ( );
          theta[joint] += degrees_per_iteration * i;
          }
    lcprintf (row, col, theta[joint]);
    }

 int select_transistor (int joint, float move)
    {
    int  transistor;

    if ( move > 0 )
          switch (joint)
             {
```

```
            case 1 : transistor = 11;
                     break;
            case 2 : transistor =  8;
                     break;
            case 3 : transistor =  6;
                     break;
            case 4 : transistor =  2;
                     break;
            case 5 : transistor = 10;
            }
      else
         switch (joint)
            {
            case 1 : transistor = 12;
                     break;
            case 2 : transistor =  7;
                     break;
            case 3 : transistor =  5;
                     break;
            case 4 : transistor =  1;
                     break;
            case 5 : transistor =  9;
            }
   return (transistor);
   }

float select_scale (int transistor)
   {
   float  scale;

   switch (transistor)
      {                                  /* Wrist Down - Up */
      case  1 : scale = 3000 /  200;
                break;
      case  2 : scale = 3650 /  200;
                break;
                                         /* Gripper Close - Open */
      case  3 : scale = 1;
                break;
      case  4 : scale = 1;
                break;
                                         /* Elbow Right - Left */
      case  5 : scale = 4150 /  180;
                break;
      case  6 : scale = 4100 /  180;
                break;
                                         /* Arm Down - Up */
      case  7 : scale = 2900 /   35;
                break;
      case  8 : scale = 2700 /   35;
                break;
                                         /* Wrist Rotate Left - Right */
      case  9 : scale = 6800 / 1080;
                break;
      case 10 : scale = 6600 / 1080;
                break;
                                         /* Arm Left - Right */
      case 11 : scale = 7100 /  360;
                break;
      case 12 : scale = 6500 /  360;
                break;
      }
   return (scale);
   }
```

APPENDIX E

POSITION AND ORIENTATION CONTROL PROCEDURES LISTING

```
#include <c:\ed's\header.c>

void position_orientation_control (f_5 theta, f_3_2 noap,
                                   int noap_row, i_4 noap_cols)
   {
   int     magnitude_ok;
   char    move;
   int     arm_row;
   i_2     arm_cols;
   int     theta_row;
   i_4     theta_cols;
   f_2     pa;
   f_4     t3;
   f_4_3   theta123;
   int     accepted;
   f_4_5   accepted_theta;
   f_5     move_theta;
   char    *prompt_msg1;
   char    *prompt_msg2;
   char    qc;
   f_5     s;
   f_5     c;
   int     i;

   dsply_position_orientation_introduction ( );
   wait_then_erase (9);
   do
      {
      dsply_pos_orient_solution (&arm_row, arm_cols,
                                 &theta_row, theta_cols);
      get_noap (noap, noap_row, noap_cols);
      magnitude_ok = calc_arm_end (noap, pa, arm_row, arm_cols);
      if ( magnitude_ok )
            {
            calc_theta_123_triples (pa, theta123,
                                    theta_row, theta_cols);
            accepted = calc_theta_45_pairs (noap, theta123,
                                            accepted_theta,
                                            theta_row, theta_cols);
            wait_then_erase (9);
            move = prompt_for_move (accepted, accepted_theta,
                                    move_theta);
            if ( move == 'Y' )
               position_orientation_move (theta, move_theta);
            noap_matrix (theta, noap, noap_row, noap_cols);
            }
         else
        lcputs (28, 19, "Arm end position not attainable");
      prompt_msg1 = "Continue with another N-O-A-P matrix (y/n)?";
      prompt_msg2 = "";
      qc = prompt_input_char (prompt_msg1, prompt_msg2);
      }
      while ( qc == 'Y' );
   wait_then_erase (8);
   }

void dsply_position_orientation_introduction ( )
   {
   int  lm;
   int  rm;

   locate (8, 0);
   lm = 16;
   rm = 16;
   mcputs (lm, rm, "          Position-Orientation Control          ");
   mcputs (lm, rm, "                                                ");
   mcputs (lm, rm, "  The movement of each of the five joints of the");
   mcputs (lm, rm, "Armatron is controlled by specifying a desired  ");
   mcputs (lm, rm, "position-orientation matrix consisting of       ");
   mcputs (lm, rm, "vectors n, o, a, and p.                         ");
```

```
    mcputs (lm, rm, "  The steps in the solution process are as      ");
    mcputs (lm, rm, "follows:                                        ");
    mcputs (lm, rm, "  1) determine the end of the arm proper from   ");
    mcputs (lm, rm, "     the desired gripper center and approach of ");
    mcputs (lm, rm, "     the wrist                                  ");
    mcputs (lm, rm, "  2) four possible triples are then evaluated   ");
    mcputs (lm, rm, "     to bring the arm proper to this postion    ");
    mcputs (lm, rm, "  3) solutions are then obtained for the wrist  ");
    mcputs (lm, rm, "     variables, if any exist                    ");
    }

void dsply_pos_orient_solution (int *arm_row, i_2 arm_cols,
                                int *theta_row, i_4 theta_cols)
    {
    int  lm;
    int  rm;

    lm = 16;
    rm = 15;
    locate (9, 0);
    mcputs (lm, rm, "      Determination of Pa Vector Components      ");
    mcputs (lm, rm, "         Pwx            Pwy            Pwz       ");
    mcputs (lm, rm, "     nnnnnnnnnn     nnnnnnnnnn     nnnnnnnnnn    ");
    mcputs (lm, rm, "         Pax            Pay            Paz       ");
    mcputs (lm, rm, "     nnnnnnnnnn     nnnnnnnnnn     nnnnnnnnnn    ");
    mcputs (lm, rm, "                                                 ");
    mcputs (lm, rm, "               Control Variable Solutions        ");
    mcputs (lm, rm, "Theta     Set 1      Set 2      Set 3      Set 4  ");
    mcputs (lm, rm, "  1    nnnnnnnnnn nnnnnnnnnn nnnnnnnnnn nnnnnnnnnn");
    mcputs (lm, rm, "  2                                              ");
    mcputs (lm, rm, "  3                                              ");
    mcputs (lm, rm, "  4                                              ");
    mcputs (lm, rm, "  5                                              ");
    mcputs (lm, rm, "el3,3                                            ");

    *arm_row = 11;
    arm_cols[0] = 21;
    arm_cols[1] = 36;
    arm_cols[2] = 51;
    *theta_row = 17;
    theta_cols[1] = 22;
    theta_cols[2] = 33;
    theta_cols[3] = 44;
    theta_cols[4] = 55;
    }

void get_noap (f_3_2 noap, int row, i_4 cols)
    {
    c_3_2_2  names;
    int      i;
    f_2      n_cross_o;
    float    difference;

    init_names (names);
    do
       {
       for (i = 0; i <= 2; i++)
          get_orientation_vector (i, names, noap, row, cols);
       n_cross_o[0] = noap[0][1]*noap[1][2] - noap[0][2]*noap[1][1];
       n_cross_o[1] = noap[0][2]*noap[1][0] - noap[0][0]*noap[1][2];
       n_cross_o[2] = noap[0][0]*noap[1][1] - noap[0][1]*noap[1][0];
       difference = fabs(magnitude (n_cross_o) - magnitude (noap[2]));
       if ( difference > tolerance )
          {
          lcputs (23, 20, "N x O does not equal A; ");
          cputs ("Re-enter Vectors N, O, and A");
          wait_then_erase (23);
          }
       }
       while ( difference > tolerance );
```

```
   get_position_vector (names, noap, row, cols);
   }

void init_names (c_3_2_2 names)
   {
   strcpy (names[0][0], "Nx");
   strcpy (names[0][1], "Ny");
   strcpy (names[0][2], "Nz");
   strcpy (names[1][0], "Ox");
   strcpy (names[1][1], "Oy");
   strcpy (names[1][2], "Oz");
   strcpy (names[2][0], "Ax");
   strcpy (names[2][1], "Ay");
   strcpy (names[2][2], "Az");
   strcpy (names[3][0], "Px");
   strcpy (names[3][1], "Py");
   strcpy (names[3][2], "Pz");
   }

void get_orientation_vector (int i, c_3_2_2 names, f_3_2 noap,
                             int row, i_4 cols)
   {
   int    j;
   float  difference;

   do
      {
      for (j = 0; j <= 2; j++)
         {
         prompt_input_noap (names[i][j], &noap[i][j]);
         erase_prompt (23);
         lcprintf (row+j, cols[i], noap[i][j]);
         }
      difference = fabs(1 - magnitude (noap[i]));
      if ( difference > tolerance )
         {
         lcputs (23, 20, "Vector Magnitude does not equal 1; ");
         cputs ("Re-enter Vector ");
     putch (names[i][1][0]);
         wait_then_erase (23);
         }
      }
      while ( difference > tolerance );
   }

void get_position_vector (c_3_2_2 names, f_3_2 noap, int row, i_4 cols)
   {
   int    j;
   float  mag_p;

   do
      {
      for (j = 0; j <= 2; j++)
         {
         prompt_input_noap (names[3][j], &noap[3][j]);
         erase_prompt (23);
         lcprintf (row+j, cols[3], noap[3][j]);
         }
      mag_p = magnitude (noap[3]);
      if ( (mag_p > a2+a3+d5) ¦ (mag_p < a2) )
         {
         lcputs (23, 20, "Specified position is outside of the arm ");
         cputs ("envelope; re-enter vector P");
         wait_then_erase (23);
         }
      }
      while ( (mag_p > a2+a3+d5) ¦ (mag_p < a2) );
   }

void prompt_input_noap (char *name, float *value)
```

```
   {
   float  new_value;

   lcputs (23, 20, "Enter N-O-A-P element ");
   cputs (name);
   cputs (" <Snnn.nnn>:  ");
   lcputs (24, 22, "<Return> only to leave value unchanged as");
   lcprintf (24, 64, *value);
   new_value = indec (23, 59);
   if ( new_value != 1000 )
      *value = new_value;
   }

int calc_arm_end (f_3_2 noap, f_2 pa, int arm_row, i_2 arm_cols)
   {
   int    i;
   f_2    pw;
   float  mag_pa;
   int    mag_ok;

   for (i = 0; i <= 2; i++)
      {
      pw[i] = d5 * noap[2][i];
      lcprintf (arm_row, arm_cols[i], pw[i]);
      }
   for (i = 0; i <= 2; i++)
      {
      pa[i] = noap[3][i] - pw[i];
      lcprintf (arm_row+2, arm_cols[i], pa[i]);
      }
   mag_pa = magnitude (pa);
   if ( (mag_pa > a2+a3) | ( (fabs(pa[0]) < tolerance) &
                             (fabs(pa[1]) < tolerance) ) )
         mag_ok = 0;
      else
         mag_ok = 1;
   return (mag_ok);
   }

void calc_theta_123_triples (f_2 pa, f_4_3 theta, int row, i_4 cols)
   {
   float  beta;
   f_4    bmt1;

   calc_theta_3 (pa, theta, row+2, cols);
   calc_beta_minus_theta_1 (pa, theta, bmt1);
   beta = calc_beta (pa);
   calc_theta_1 (beta, bmt1, theta, row, cols);
   calc_theta_2 (pa, theta, row+1, cols);
   }

void calc_theta_3 (f_2 pa, f_4_3 theta, int row, i_4 cols)
   {
   float  c3;
   float  s3;
   int    i;

   c3 = (square (magnitude (pa)) - square (a2) - square (a3))
      / (2 * a2 * a3);
   s3 = sqrt(1 - square (c3));
   if ( fabs(c3) > tolerance )
         {
         theta[1][3] = atan( s3 / c3);
         theta[3][3] = atan(-s3 / c3);
                                        /* adjust atan for cos < 0 */
         if ( c3 < 0 )
            {
                                        /* c3<0, s3>0 => 4th->2nd quad */
            theta[1][3] += pi;
                                        /* c3<0, s3<0 => 1st->3rd quad */
            theta[3][3] -= pi;
            }
         }
      else
         {
         theta[1][3] =  pi / 2;
         theta[3][3] = -pi / 2;
         }
                                          /* 2 copies => 4 triples */
   theta[2][3] = theta[1][3];
   theta[4][3] = theta[3][3];

   for (i = 1; i <= 4; i++)
      lcprintf (row, cols[i], theta[i][3] * 180/pi);
   }


void calc_beta_minus_theta_1 (f_2 pa, f_4_3 theta, f_4 bmt1)
   {
   int    i;
   float  sbmt1;
   float  cbmt1;
                                          /* for 2 pairs:  (1,2) & (3,4) */
   for (i = 1; i <= 3; i = i + 2)
      {
      sbmt1 = a3 * sin(theta[i][3])
            / sqrt(square (pa[0]) + square (pa[1]));
      cbmt1 = sqrt(1 - square (sbmt1));
      if ( fabs(cbmt1) > tolerance )
            {
            bmt1[i  ] = atan(sbmt1 /  cbmt1);
            bmt1[i+1] = atan(sbmt1 / -cbmt1);
                                          /* adjust 1 due to sine */
            bmt1[i+1] = bmt1[i+1] + pi;
            }
         else
            {
            if ( sbmt1 > 0 )
                bmt1[i] =  pi / 2;
              else
                bmt1[i] = -pi / 2;
            bmt1[i+1] = bmt1[i];
            }
      }
   }

float calc_beta (f_2 pa)
   {
   int    sign_cos_beta;
   int    sign_sin_beta;
   float  beta;

   sign_cos_beta = sign (pa[0]);
   sign_sin_beta = sign (pa[1]);
   if ( fabs(pa[0]) > tolerance )
         {
         beta = atan (pa[1] / pa[0]);
                                          /* adjust atan for cos < 0 */
         if ( sign_cos_beta == -1 )
            beta += pi;
         }
      else
                                          /* cos = 0, + sin => +90 deg */
         if ( sign_sin_beta == +1 )
               beta =  pi / 2;
                                          /* cos = 0, - sin => -90 deg */
            else
               beta = -pi / 2;
   return (beta);
   }
```

```
void calc_theta_1 (float beta, f_4 bmt1, f_4_3 theta, int row, i_4 cols)
   {
   int  i;

   for (i = 1; i <= 4; i++)
      {
                                             /* t1 = beta - (beta - t1) */
      theta[i][1] = beta - bmt1[i];
                                             /* adjust if >  180 degrees */
      if ( theta[i][1] >  pi )
         theta[i][1] -= 2 * pi;
                                             /* adjust if < -180 degrees */
      if ( theta[i][1] < -pi )
         theta[i][1] += 2 * pi;
      lcprintf (row, cols[i], theta[i][1] * 180/pi);
      }
   }

void calc_theta_2 (f_2 pa, f_4_3 theta, int row, i_4 cols)
   {
   float  c3;
   int    i;
   float  c2;
   float  s2;
                                             /* cos(theta 3) is constant */
   c3 = cos(theta[1][3]);
   for (i = 1; i <= 4; i++)
      {
      c2 = ( pa[0]*cos(theta[i][1]) + pa[1]*sin(theta[i][1]) )
         / (a3*c3 + a2);
      s2 = pa[2] / (a3*c3 + a2);
      if ( fabs(c2) > tolerance )
            {
            theta[i][2] = atan(s2 / c2);
                                             /* adjust atan for cos < 0 */
            if ( c2 < 0 )
                                             /* -c, +s => 2nd quad from 4th */
               if ( s2 >= 0 )
                     theta[i][2] += pi;
                                             /* -c, -s => 3rd quad from 1st */
                  else
                     theta[i][2] -= pi;
            }
         else
            if ( s2 > 0 )
                  theta[i][2] =  pi / 2;
               else
                  theta[i][2] = -pi / 2;
      lcprintf (row, cols[i], theta[i][2] * 180/pi);
      }
   }

int calc_theta_45_pairs (f_3_2 noap, f_4_3 theta, f_4_5 accepted_theta,
                         int row, i_4 cols)
   {
   f_3    c;
   f_3    s;
   int    i;
   int    j;
   int    atc;
   float  el_3_3;

   atc = 0;
   for (i = 1; i <= 4; i++)
      {
      for (j = 1; j <= 3; j++)
         {
         c[j] = cos(theta[i][j]);
         s[j] = sin(theta[i][j]);
```

```
            }
        el_3_3 = noap[2][0] * (c[1]*c[2]*s[3] + s[1]*c[3])
               + noap[2][1] * (s[1]*c[2]*s[3] - c[1]*c[3])
               + noap[2][2] *  s[2]*s[3];
        lcprintf (row+5, cols[i], el_3_3);
        if ( fabs(el_3_3) < tolerance )
              {
              atc++;
              for (j = 1; j <= 3; j++)
                 accepted_theta[atc][j] = theta[i][j];
              accepted_theta[atc][4] = calc_theta_4 (noap[2], s, c);
              lcprintf (row+3, cols[i], accepted_theta[atc][4] * 180/pi);
              accepted_theta[atc][5] = calc_theta_5 (noap[0], noap[1],
                                                       s, c);
              lcprintf (row+4, cols[i], accepted_theta[atc][5] * 180/pi);
              }
           else
              {
             lcputs (row+3, cols[i], "     No    ");
             lcputs (row+4, cols[i], "  Solution");
              }
        }
    return (atc);
    }

float calc_theta_4 (f_2 a, f_3 s, f_3 c)
    {
    float  c4;
    float  s4;
    float  theta4;

    c4 = a[0]*c[1]*s[2] + a[1]*s[1]*s[2] - a[2]*c[2];
    s4 = a[0] * (c[1]*c[2]*c[3] - s[1]*s[3])
       + a[1] * (s[1]*c[2]*c[3] + c[1]*s[3]) + a[2]*s[2]*c[3];
    if ( fabs(c4) > tolerance )
          {
          theta4 = atan(s4 / c4);
                                          /* adjust atan for cos < 0 */
          if ( c4 < 0 )
                                          /* -c, +s => 2nd quad from 4th */
             if (s4 >= 0)
                   theta4 += pi;
                                          /* -c, -s => 3rd quad from 1st */
                else
                   theta4 -= pi;
                                          /* conv for bounds compliance */
          if ( theta4 < -170 * pi/180 )
             theta4 += 2 * pi;
          }
       else
                                          /* cos = 0, +sin => +90 deg */
          if ( s4 > 0 )
                theta4 =  pi / 2;
                                          /* cos = 0, -sin => -90 deg */
             else
                theta4 = -pi / 2;
    return (theta4);
    }

float calc_theta_5 (f_2 n, f_2 o, f_3 s, f_3 c)
    {
    float  c5;
    float  s5;
    float  theta5;

    c5 = o[0]*(c[1]*c[2]*s[3] + s[1]*c[3])
       + o[1]*(s[1]*c[2]*s[3] - c[1]*c[3]) + o[2]*s[2]*s[3];
    s5 = n[0]*(c[1]*c[2]*s[3] + s[1]*c[3])
       + n[1]*(s[1]*c[2]*s[3] - c[1]*c[3]) + n[2]*s[2]*s[3];
```

```
   if ( fabs(c5) > tolerance )
         {
         theta5 = atan(s5 / c5);
                                       /* adjust atan for cos < 0 */
         if ( c5 < 0 )
                                       /* -c, +s => 2nd quad from 4th */
            if (s5 >= 0)
                  theta5 += pi;
                                       /* -c, -s => 3rd quad from 1st */
               else
                  theta5 -= pi;
         }
      else
                                       /* cos = 0, +sin => +90 deg */
         if ( s5 > 0 )
               theta5 =  pi / 2;
                                       /* cos = 0, -sin => -90 deg */
            else
               theta5 = -pi / 2;
   return (theta5);
   }

char prompt_for_move (int accepted, f_4_5 accepted_theta,
                      f_5 move_theta)
   {
   int   row;
   i_8   cols;
   int   inbounds;
   int   i;
   i_4   out;
   int   j;
   i_4   inbounds_index;
   char  move;

   dsply_prompt_for_move (&row, cols);
   inbounds = 0;
   for (i = 1; i <= accepted; i++)
      {
      out[i] = 0;
      for (j = 1; j <= 5; j++)
         {
         accepted_theta[i][j] *= (180 / pi);
         lcprintf (row+j-1, cols[i], accepted_theta[i][j]);
         if ( (accepted_theta[i][j] >= min_constraint (j))
            & (accepted_theta[i][j] <= max_constraint (j)) )
               lcputs (row+j-1, cols[i+4]+2, "In");
            else
               {
               lcputs (row+j-1, cols[i+4]+1, "Out");
               out[i]++;
               }
         }
      if ( out[i] == 0 )
         {
         inbounds++;
         inbounds_index[inbounds] = i;
         }
      }
   switch (inbounds)
      {
      case 0 : lcputs (20, 20, "No Solution is Obtainable");
               move = 'N';
               break;
      case 1 : move = one_solution (inbounds_index[1], accepted_theta,
                                    move_theta);
               break;
      default: move = multiple_solutions (accepted, accepted_theta,
                                          inbounds, inbounds_index,
                                          out, move_theta);
      }
   erase_prompt (22);
   return (move);
   }

void dsply_prompt_for_move (int *row, i_8 cols)
   {
   locate (9, 0);
   cputs ("                                    Solutions");
   cputs ("                                   ");
   cputs ("Theta            Bounds             Bounds              ");
   cputs ("Bounds             Bounds");
   cputs ("   1                                                    ");
   cputs ("                         ");
   cputs ("   2                                                    ");
   cputs ("                         ");
   cputs ("   3                                                    ");
   cputs ("                         ");
   cputs ("   4                                                    ");
   cputs ("                         ");
   cputs ("   5                                                    ");
   cputs ("                         ");

   *row = 11;
   cols[1] =  6;
   cols[2] = 25;
   cols[3] = 44;
   cols[4] = 63;
   cols[5] = 18;
   cols[6] = 37;
   cols[7] = 56;
   cols[8] = 75;
   }

float min_constraint (int joint)
   {
   float  minimum;

   switch (joint)
      {
      case 1 : minimum = -360;
               break;
      case 2 : minimum =   -5;
               break;
      case 3 : minimum =  -90;
               break;
      case 4 : minimum =  -10;
               break;
      case 5 : minimum = -360;
               break;
      }
   return (minimum);
   }

float max_constraint (int joint)
   {
   float  maximum;

   switch (joint)
      {
      case 1 : maximum =  360;
               break;
      case 2 : maximum =   30;
               break;
      case 3 : maximum =   90;
               break;
      case 4 : maximum =  190;
               break;
      case 5 : maximum =  360;
               break;
      }
```

```
    return (maximum);
    }

char one_solution (int index, f_4_5 accepted_theta, f_5 move_theta)
    {
    char  move;
    int   j;

    locate (23, 20);
    cprintf ("Solution %d is obtainable", index);
    lcputs (24, 20, "Perform move? (y/n)");
    locate (24, 42);
    move = toupper(getch( ));
    if (move == 'Y')
       for (j = 1; j <= 5; j++)
          move_theta[j] = accepted_theta[index][j];
    return (move);
    }

char multiple_solutions (int accepted, f_4_5 accepted_theta,
                         int inbounds, i_4 inbounds_index, i_4 out,
                         f_5 move_theta)
    {
    int   i;
    char  *prompt;
    int   cont;
    int   set;
    char  move;
    int   j;

    locate (22, 20);
    cprintf ("Solutions %d", inbounds_index[1]);
    for (i = 2; i <= inbounds-1; i++)
       cprintf (", %d", inbounds_index[i]);
    cprintf (" and %d are obtainable.", inbounds_index[inbounds]);
    prompt = "Select set for a move or 0 to abort:";
    do
       {
       set = prompt_input_digit (prompt);
       if ( set == 0 )
             cont = 0;
          else
             if ( (set <= accepted) & (out[set] == 0) )
                   cont = 0;
                else
                   cont = 1;
       }
       while ( cont );
    locate (24, 20);
    cprintf ("Solution %d has been selected", set);
    if ( set == 0 )
          move = 'N';
       else
          {
          move = 'Y';
          for (j = 1; j <= 5; j++)
             move_theta[j] = accepted_theta[set][j];
          }
    return (move);
    }

void position_orientation_move (f_5 theta, f_5 move_theta)
    {
    int  row;
    i_2  cols;
    int  i;
    int  interrupt_count;

    dsply_pos_orient_move (&row, cols);
    for (i = 1; i <= 5; i++)
```

```
        {
        lcprintf (row+i-1, cols[0], theta[i]);
        lcprintf (row+i-1, cols[1], move_theta[i]);
        }
    wait_then_erase (24);
    interrupt_count = 0;
    for (i = 1; i <= 5; i++)
        {
        perform_move (i, theta, move_theta[i], row+i-1, cols[0]);
        if ( fabs(theta[i] - move_theta[i]) < tolerance )
                lcputs (row+i-1, cols[2], "Yes");
            else
                {
                lcputs (row+i-1, cols[2], " No");
                interrupt_count++;
                }
        }
    if ( interrupt_count == 0 )
            {
            lcputs (23, 15, "Motion completed; ");
            cputs ("Position-Orientation achieved");
            }
        else
            {
            lcputs (23, 15, "Some motion interrupted; ");
            cputs ("Position-Orientation not achieved");
            }
    wait_then_erase (9);
    }

void dsply_pos_orient_move (int *row, i_2 cols)
    {
    int  lm;
    int  rm;

    lm = 20;
    rm = 20;
    locate (16, 0);
    mcputs (lm, rm, "              Moving               ");
    mcputs (lm, rm, "Theta    Current     Desired   Completed");
    mcputs (lm, rm, "  1                                    ");
    mcputs (lm, rm, "  2                                    ");
    mcputs (lm, rm, "  3                                    ");
    mcputs (lm, rm, "  4                                    ");
    mcputs (lm, rm, "  5                                    ");

    *row    = 18;
    cols[0] = 27;
    cols[1] = 39;
    cols[2] = 54;
    }
```

APPENDIX F

VELOCITY CONTROL PROCEDURES LISTING

```
#include <c:\ed's\header.c>

void velocity_control (f_5 original_theta, f_3_2 original_noap,
                       int row, i_4 cols)
   {
   int    i;
   int    j;
   f_5    theta;
   f_3_2  noap;
   char   *prompt_msg1;
   char   *prompt_msg2;
   char   qc;
   int    opt;
   f_5    dtheta;
   f_6    delta_trans_rot;

   for (i = 1; i <= 5; i++)
      {
      theta[i] = original_theta[i];
      dtheta[i] = 0;
      }
   for (i = 1; i <= 6; i++)
      delta_trans_rot[i] = 0;
   for (i = 0; i <= 3; i++)
      for (j = 0; j <= 2; j++)
         noap[i][j] = original_noap[i][j];

   dsply_velocity_introduction ( );
   wait_then_erase (9);
   dsply_vc_selection ( );
   while ((opt = get_option(3)) != 0)
      {
      prompt_msg1 = "Enter New Theta Values? (Y/N)";
      prompt_msg2 = "(<N> = continue with previous values)";
      qc = prompt_input_char (prompt_msg1, prompt_msg2);
      if (qc == 'Y')
         {
         get_theta (theta, row, cols[4]);
         noap_matrix (theta, noap, row, cols);
         }
      wait_then_erase (9);
      switch (opt)
         {
         case 1 : for_sol_via_jac (theta, dtheta);
                  break;
         case 2 : rev_sol_via_ij  (theta, delta_trans_rot);
                  break;
         case 3 : rev_sol_via_deriv (theta, noap, delta_trans_rot);
                  break;
         }
      dsply_vc_selection ( );
      }
   wait_then_erase (8);
   noap_matrix (original_theta, original_noap, row, cols);
   }

void dsply_velocity_introduction ( )
   {
   int  lm;
   int  rm;

   locate (8, 0);
   lm = 16;
   rm = 16;
   mcputs (lm, rm, "                 Velocity Control                  ");
   mcputs (lm, rm, "  This section calculates the velocities of      ");
   mcputs (lm, rm, "the end coordinate frame or the joint variables.");
   mcputs (lm, rm, "  Options:                                        ");
   mcputs (lm, rm, "  1) Forward Solutions via Jacobian               ");
   mcputs (lm, rm, "    -the end coordinate frame rates resulting     ");
   mcputs (lm, rm, "     from a given set of joint rates are found  ");
   mcputs (lm, rm, "  2) Reverse Solutions via Inverse Jacobian     ");
   mcputs (lm, rm, "    -the joint rates resulting from a given set ");
   mcputs (lm, rm, "     of coordinate frame rates are obtained     ");
   mcputs (lm, rm, "     using matrix algebra                       ");
   mcputs (lm, rm, "  3) Reverse Solutions via Derivatives          ");
   mcputs (lm, rm, "    -the joint rates resulting from a given set ");
   mcputs (lm, rm, "     of coordinate frame rates are obtained from");
   mcputs (lm, rm, "     derivatives of the position-orientation    ");
   mcputs (lm, rm, "     equations                                  ");
   }

void dsply_vc_selection ()
   {
   int  r;
   int  c;

   r = 10;
   c = 20;
   locate (9, 0);
   lcputs (r,   c, "Solution Options                           ");
   lcputs (r+2, c, "1: Forward Solutions via Jacobian Matrix ");
   lcputs (r+4, c, "2: Reverse Solutions via Inverse Jacobian");
   lcputs (r+6, c, "3: Reverse Solutions via Derivatives     ");
   lcputs (r+8, c, "0: Terminate Velocity Control            ");
   }

void get_theta (f_5 theta, int row, int col)
   {
   int    i;
   float  value;
   char   *prompt_msg;

   for (i = 1; i <= 5; i++)
      {
      prompt_msg = "Enter value for Theta";
      value = prompt_input_fixed (prompt_msg, i);
      if (value != 1000)
         theta[i] = value;
      lcprintf (row+i-2, col, theta[i]);
      }
   }

void for_sol_via_jac (f_5 theta, f_5 dtheta)
   {
   int    i;
   f_5    s;
   f_5    c;
   int    row;
   i_6    cols;
   f_6_5  jacobian;
   f_6    drate;
   char   query_ch;

   sin_cos (theta, s, c);
   do
      {
      dsply_jacobian (&row, cols);
      calc_jacobian (s, c, jacobian, row, cols);
      get_delta_theta (dtheta, row, cols[6]);
      calc_list_rates (drate, dtheta, jacobian, row, cols[5]);
      query_ch = cont ("new Jacobian and/or theta rates");
      }
      while ( query_ch == 'Y' );
   }

void dsply_jacobian (int *row, i_6 cols)
   {
   locate (9, 0);
   mcputs (23, 23, "Forward Solutions via the Jacobian");
   cputs (" Delta Rates                         Jacobian  ");
   cputs ("                                 ");
   cputs ("|          | |                                 ");
   cputs ("                     | |        |");
   cputs ("|          | |                                 ");
   cputs ("                     | |        |");
   cputs ("|          |=|                                 ");
   cputs ("                     |X|        |");
   cputs ("|          | |                                 ");
   cputs ("                     | |        |");
   cputs ("|          | |                                 ");
   cputs ("                     | |        |");
   cputs ("|          | |                                 ");
   cputs ("                     |           ");

   *row   = 11;
   cols[0] = 14;
   cols[1] = 25;
   cols[2] = 36;
   cols[3] = 47;
   cols[4] = 58;
   cols[5] =  1;
   cols[6] = 71;
   }

void calc_jacobian (f_5 s, f_5 c, f_6_5 jacobian, int row, i_5 cols)
   {
   f_12  f;
   int   i;
   int   j;
                              /* Misc Factors, Columns 1-4 */
   f[0] = d5*s[4] + a3;
   f[1] = f[0]*c[3] + a2;
   f[2] = f[1]*c[2] + d5*s[2]*c[4];
   f[3] = f[0]*s[3];
                              /* Jacobian Column 1 */
   jacobian[1][1] = -f[2]*s[1] - f[3]*c[1];
   jacobian[2][1] =  f[2]*c[1] - f[3]*s[1];
   jacobian[3][1] =  0;
   jacobian[4][1] =  0;
   jacobian[5][1] =  0;
   jacobian[6][1] =  1;
                              /* Misc Factor, Column 2 */
   f[4] = f[1]*s[2] - d5*c[2]*c[4];
                              /* Jacobian Column 2 */
   jacobian[1][2] = -f[4]*c[1];
   jacobian[2][2] = -f[4]*s[1];
   jacobian[3][2] =  f[2];
   jacobian[4][2] =  s[1];
   jacobian[5][2] = -c[1];
   jacobian[6][2] =  0;
                              /* Misc Factors, Columns 3 & 4 */
   f[5] = c[2]*s[3];
   f[6] = f[5]*c[1] + c[3]*s[1];
   f[7] = f[5]*s[1] - c[3]*c[1];
                              /* Jacobian Column 3 */
   jacobian[1][3] = -f[0]*f[6];
   jacobian[2][3] = -f[0]*f[7];
   jacobian[3][3] = -f[3]*s[2];
   jacobian[4][3] = -c[1]*s[2];
   jacobian[5][3] = -s[1]*s[2];
   jacobian[6][3] =  c[2];
                              /* Misc Factors, Columns 4 & 5 */
   f[8]  = c[1]*c[2]*c[3] - s[1]*s[3];
   f[9]  = s[1]*c[2]*c[3] + c[1]*s[3];
   f[10] = c[1]*s[2];
   f[11] = s[1]*s[2];
   f[12] = s[2]*c[3];
                              /* Jacobian Column 4 */
```

```
    jacobian[1][4] = d5*(f[ 8]*c[4] - f[10]*s[4]);
    jacobian[2][4] = d5*(f[ 9]*c[4] - f[11]*s[4]);
    jacobian[3][4] = d5*(f[12]*c[4] + c[ 2]*s[4]);
    jacobian[4][4] = f[6];
    jacobian[5][4] = f[7];
    jacobian[6][4] = s[2]*s[3];
                                        /* Jacobian Column 5 */
    jacobian[1][5] = 0;
    jacobian[2][5] = 0;
    jacobian[3][5] = 0;
    jacobian[4][5] = f[ 8]*s[4] + f[10]*c[4];
    jacobian[5][5] = f[ 9]*s[4] + f[11]*c[4];
    jacobian[6][5] = f[12]*s[4] - c[ 2]*c[4];

    for (j = 1; j <= 5; j++)
       for (i = 1; i <= 6; i++)
          lcprintf (row+i-1, cols[j-1], jacobian[i][j]);
    }

void get_delta_theta (f_5 dtheta, int r, int c)
    {
    int    i;
    float  value;
    char   *prompt_msg;

    for (i = 1; i <= 5; i++)
       {
       locate (23, 20);
       cprintf ("Enter value for Delta Theta %d <Snnn.nnn>:  ", i);
       lcputs (24, 22, "<Return> only to leave unchanged as");
       lcprintf (24, 58, dtheta[i]);
       value = indec (23, 63);
       if (value != 1000)
          dtheta[i] = value;
       erase_prompt (23);
       lcprintf8 (r+i-1, c, dtheta[i]);
       }
    }

void calc_list_rates (f_6 drate, f_5 dtheta, f_6_5 jacobian,
                      int r, int c)
    {
    int  i;
    int  j;

    for (i = 1; i <= 6; i++)
       {
       drate[i] = 0;
       for (j = 1; j <= 5; j++)
          drate[i] += jacobian[i][j] * dtheta[j];
       lcprintf (r+i-1, c, drate[i]);
       }
    }

char cont (char *msg)
    {
    char  answer;

    save_screen ( );
    lcputs (23, 20, "Continue with ");
    cputs (msg);
    lcputs (24, 20, "but same Theta values? (y/n)");
    locate (24, 50);
    answer = toupper ( getch ( ) );
    erase_prompt (9);
    return (answer);
    }

void rev_sol_via_ij  (f_5 theta, f_6 delta_trans_rot)
    {
```

```
   int    i;
   f_5    s;
   f_5    c;
   int    row;
   i_5    cols;
   f_6_5  jacobian;
   i_6    used;
   int    t;
   f_6_5  jacobian_reduced;
   f_6    delta_tr_reduced;
   f_5    delta_theta;
   int    ic;
   char   qc;

   sin_cos (theta, s, c);
   do
      {
      dsply_rsvij_jacobian (&row, cols);
      calc_jacobian (s, c, jacobian, row, cols);
      t = get_required_rates (delta_trans_rot, jacobian, used,
                              jacobian_reduced, delta_tr_reduced,
                              row, cols);
      wait_then_erase (10);
      if ( (t > 0) & (t < 6) )
         {
         lcputs (10, 30, "Under-Determined Case");
         ic = under_determined_case (t, jacobian_reduced,
                                     delta_tr_reduced, delta_theta);
         }
      if ( t == 6 )
         {
         lcputs (10, 30, "Over-Determined Case");
         ic = over_determined_case (jacobian, delta_trans_rot,
                                    delta_theta);
         }
      if ( (t > 0) & (!ic) )
         list_input_output (delta_trans_rot, used, delta_theta);
      qc = cont ("different Jacobian and/or rates");
      }
      while ( qc == 'Y' );
   }

void dsply_rsvij_jacobian (int *row, i_5 cols)
   {
   locate (9, 0);
   mcputs (21, 21, "Reverse Solutions via Inverse Jacobian");
   mcputs (21, 21, "                                      ");
   cputs (" Delta Rates                       Jacobian ");
   cputs ("                                    ");
   cputs (" ¦          ¦   ¦                                 ");
   cputs ("                     ¦ ¦d T1¦ ");
   cputs (" ¦          ¦   ¦                                 ");
   cputs ("                     ¦ ¦d T2¦ ");
   cputs (" ¦          ¦ = ¦                                 ");
   cputs ("                     ¦X¦d T3¦ ");
   cputs (" ¦          ¦   ¦                                 ");
   cputs ("                     ¦ ¦d T4¦ ");
   cputs (" ¦          ¦   ¦                                 ");
   cputs ("                     ¦ ¦d T5¦ ");
   cputs (" ¦          ¦   ¦                                 ");
   cputs ("                     ¦        ");

   *row    = 12;
   cols[0] = 17;
   cols[1] = 28;
   cols[2] = 39;
   cols[3] = 50;
   cols[4] = 61;
   cols[5] =  2;
   }
```

```
int  get_required_rates (f_6 delta_trans_rot, f_6_5 jacobian, i_6 used,
                         f_6_5 jacobian_reduced, f_6 delta_tr_reduced,
                         int row, i_5 cols)
   {
   c_6_9  rate_lbl;
   int    i;
   int    j;
   char   *line1;
   char   qc;
   float  value;
   int    total;

   total = 0;
   strcpy(rate_lbl[1], "transl x");
   strcpy(rate_lbl[2], "transl y");
   strcpy(rate_lbl[3], "transl z");
   strcpy(rate_lbl[4], "rotate x");
   strcpy(rate_lbl[5], "rotate y");
   strcpy(rate_lbl[6], "rotate z");

   for (i = 1; i <= 6; i++)
      {
      lcputs (23, 20, "Make d ");
      cputs (rate_lbl[i]);
      cputs (" a command variable? (y/n) ");
      locate (23, 63);
      qc = toupper(getch( ));
      erase_prompt (23);
      if ( qc != 'N' )
            {
            total++;
            used[i] = 1;
            for (j = 1; j <= 5; j++)
               jacobian_reduced[total][j] = jacobian[i][j];
            lcputs (23, 20, "Enter d ");
            cputs (rate_lbl[i]);
            cputs (" :  ");
            lcputs (24, 22, "<Return> only to leave value unchanged as");
            lcprintf (24, 64, delta_trans_rot[i]);
            value = indec (23, 40);
            if ( value != 1000 )
               delta_trans_rot[i] = value;
            erase_prompt (23);
            lcprintf (row+i-1, cols[5], delta_trans_rot[i]);
            delta_tr_reduced[total] = delta_trans_rot[i];
            }
         else
            {
            used[i] = 0;
            lcputs (row+i-1, cols[5], "   unused ");
            }
      }
   return (total);
   }

int over_determined_case (f_6_5 jacobian, f_6 delta_trans_rot,
                          f_5 delta_theta)
   {
   f_5_5  m;
   f_5    v;
   f_5    y;
   char   *subhead;
   int    inconsistent;

   subhead = "1.  M = (J Transpose) * J";
   matrix_by_matrix (6, jacobian, m, subhead);
   subhead = "2.  V = (J Transpose) * T/R Rates";
   matrix_by_vector (jacobian, 6, delta_trans_rot, v, subhead);
   subhead = "3.  Solve M * Theta Rates = V";
   inconsistent = solve_simul_eqns_myv (m, v, 5, delta_theta, subhead);
   return (inconsistent);
   }

int under_determined_case (int total, f_6_5 jacobian_reduced,
                           f_6 delta_tr_reduced, f_5 delta_theta)
   {
   int     i;
   f_5_5   m;
   f_5     dtr_5;
   f_5     y;
   char    *subhead;
   int     inconsistent;

   subhead = "1.  M = J Reduced * (J Reduced Transpose)";
   matrix_by_matrix (total, jacobian_reduced, m, subhead);
   for (i = 1; i <= total; i++)
      dtr_5[i] = delta_tr_reduced[i];
   subhead = "2.  Solve M * Y = d trans/rot rates";
   inconsistent = solve_simul_eqns_myv (m, dtr_5, total, y, subhead);
   if ( inconsistent == 0 )
      {
      subhead = "3.  d Theta = (J Reduced Transpose) * Y";
      matrix_by_vector (jacobian_reduced, total, y, delta_theta,
                        subhead);
      }
   return (inconsistent);
   }

void matrix_by_matrix (int total, f_6_5 mparm, f_5_5 m, char *subhead)
   {
   int  i;
   int  j;
   int  k;
   int  size;
   int  length;
   int  row;
   i_5  cols;

   dsply_m (subhead, &row, cols);
   if ( total == 6 )
         {
         size = 5;
         length = 6;
         }
      else
         {
         size = total;
         length = 5;
         }
   for (i = 1; i <= size; i++)
      for (j = 1; j <= size; j++)
         {
         m[i][j] = 0;
         for (k = 1; k <= length; k++)
            if ( total == 6 )
                  m[i][j] += mparm[k][i] * mparm[k][j];
               else
                  m[i][j] += mparm[i][k] * mparm[j][k];
         lcprintf (row+i-1, cols[j], m[i][j]);
         }
   wait_then_erase (11);
   }

void dsply_m (char *subhead, int *row, i_5 cols)
   {
   lcputs (11, 20, subhead);
   locate (13, 0);
   cputs ("          |                              ");
   cputs ("                            |         ");
   cputs ("           ¦                                  ");
   cputs ("                         ¦          ");
   cputs ("           ¦                                  ");
   cputs ("                         ¦          ");
   cputs ("           ¦                                  ");
   cputs ("                         ¦          ");
   cputs ("           ¦                                  ");
   cputs ("                         ¦          ");
   *row = 13;
   cols[1] = 11;
   cols[2] = 23;
   cols[3] = 35;
   cols[4] = 47;
   cols[5] = 59;
   }

void matrix_by_vector (f_6_5 jac_parm, int total, f_5 vec, f_5 result,
                       char *subhead)
   {
   int  i;
   int  j;
   int  row;
   i_7  cols;

   dsply_m_by_v (subhead, &row, cols);
   for (i = 1; i <= total; i++)
      for (j = 1; j <= 5; j++)
         lcprintf8 (row+j-1, cols[i], jac_parm[i][j]);
   for (i = 1; i <= total; i++)
      lcprintf8 (row+i-1, cols[7], vec[i]);
   for (i = 1; i <= 5; i++)
      {
      result[i] = 0;
      for (j = 1; j <= total; j++)
         result[i] += jac_parm[j][i] * vec[j];
      lcprintf (row+i-1, cols[0], result[i]);
      }
   wait_then_erase (11);
   }

void dsply_m_by_v (char *subhead, int *row, i_7 cols)
   {
   lcputs (11, 20, subhead);
   locate (13, 0);
   cputs ("¦          ¦  ¦                         ");
   cputs ("                          ¦ ¦        ¦");
   cputs ("¦          ¦  ¦                         ");
   cputs ("                          ¦ ¦        ¦");
   cputs ("¦          ¦ =¦                         ");
   cputs ("                          ¦X¦        ¦");
   cputs ("¦          ¦  ¦                         ");
   cputs ("                          ¦ ¦        ¦");
   cputs ("¦          ¦  ¦                         ");
   cputs ("                          ¦ ¦        ¦");
   *row = 13;
   cols[0] =  1;
   cols[1] = 15;
   cols[2] = 24;
   cols[3] = 33;
   cols[4] = 42;
   cols[5] = 51;
   cols[6] = 60;
   cols[7] = 71;
   }

void list_input_output (f_6 delta_tran_rot, i_6 used, f_5 delta_theta)
   {
   int  i;
   i_1  row;
   int  col;
   dsply_in_out (row, &col);
   for (i = 1; i <= 6; i++)
      if ( used[i] )
         lcprintf (row[0]+i-1, col, delta_tran_rot[i]);
   for (i = 1; i <= 5; i++)
      lcprintf (row[1]+i-1, col, delta_theta[i]);
   }

void dsply_in_out (i_1 row, int *col)
   {
   int  lm;
   int  rm;

   lm = 18;
   rm = 17;
   locate (11, 0);
   mcputs (lm, rm, "Input: Delta Translational & Rotational Rates");
   mcputs (lm, rm, "           ¦dtx¦     ¦   unused ¦            ");
   mcputs (lm, rm, "           ¦dty¦     ¦   unused ¦            ");
   mcputs (lm, rm, "           ¦dtz¦  =  ¦   unused ¦            ");
   mcputs (lm, rm, "           ¦drx¦     ¦   unused ¦            ");
   mcputs (lm, rm, "           ¦dry¦     ¦   unused ¦            ");
   mcputs (lm, rm, "           ¦drz¦     ¦   unused ¦            ");
   mcputs (lm, rm, "         Output:  Delta Theta Rates          ");
   mcputs (lm, rm, "           ¦dT1¦     ¦          ¦            ");
   mcputs (lm, rm, "           ¦dT2¦     ¦          ¦            ");
   mcputs (lm, rm, "           ¦dT3¦  =  ¦          ¦            ");
   mcputs (lm, rm, "           ¦dT4¦     ¦          ¦            ");
   mcputs (lm, rm, "           ¦dT5¦     ¦          ¦            ");

   row[0] = 12;
   row[1] = 19;
   *col   = 40;
   }

int solve_simul_eqns_myv (f_5_5 m, f_5 v, int n, f_5 y, char *subhead)
   {
   int  i;
   int  j;
   int  k;
   int  row;
   i_7  cols;
   int  inconsistent;

   dsply_soln_myv (subhead, &row, cols);
   for (j = 1; j <= n; j++)
      for (i = 1; i <= n; i++)
         lcprintf (row+i-1, cols[j], m[i][j]);
   for (i = 1; i <= n; i++)
      {
      locate (row+i-1, cols[6]);
      cprintf ("Y(%d)", i);
      lcprintf (row+i-1, cols[7], v[i]);
      }

   k = 1;
   inconsistent = 0;
   while ( ( k <= n ) & ( !inconsistent) )
      {
      if ( fabs(m[k][k]) < tolerance )
         inconsistent = interchange_rows (m, v, k, n, row, cols);
      if ( !inconsistent )
         {
         zero_column_k (m, v, k, n, row, cols);
         k++;
         }
      }

   if ( !inconsistent )
```

```
         solve_y_vector (m, v, n, y, row, cols);
      else
         {
         locate (22, 20);
         cputs ("Equations Inconsistent; No Solution");
         }
   wait_then_erase (11);
   return (inconsistent);
   }

void dsply_soln_myv (char *subhead, int *row, i_7 cols)
   {
   lcputs (11, 20, subhead);
   locate (13, 0);
   cputs ("¦                                             ");
   cputs ("             ¦   ¦    ¦   ¦          ¦");
   cputs ("¦                                             ");
   cputs ("             ¦   ¦    ¦   ¦          ¦");
   cputs ("¦                                             ");
   cputs ("             ¦ X ¦    ¦ = ¦          ¦");
   cputs ("¦                                             ");
   cputs ("             ¦   ¦    ¦   ¦          ¦");
   cputs ("¦                                             ");
   cputs ("             ¦   ¦    ¦   ¦          ¦");
   *row = 13;
   cols[1] =  1;
   cols[2] = 12;
   cols[3] = 23;
   cols[4] = 34;
   cols[5] = 45;
   cols[6] = 60;
   cols[7] = 69;
   }

int interchange_rows (f_5_5 m, f_5 v, int k, int n, int row, i_7 cols)
   {
   int    i;
   int    j;
   float  temp;
   int    inconsistent;

   i = k + 1;
   while ( ( fabs(m[i][k]) < tolerance ) & ( i <= n ) )
      i++;
   if ( i <= n)
         {
         for (j = k; j <= n; j++)
            {
            temp = m[k][j];
            m[k][j] = m[i][j];
            lcprintf (row+k-1, cols[j], m[k][j]);
            m[i][j] = temp;
            lcprintf (row+i-1, cols[j], m[i][j]);
            pause (500);
            }
         temp = v[k];
         v[k] = v[i];
         lcprintf (row+k-1, cols[6], v[k]);
         v[i] = temp;
         lcprintf (row+i-1, cols[7], v[i]);
         pause (500);

         inconsistent = 0;
         }
      else
         inconsistent = 1;
   return (inconsistent);
   }

void zero_column_k (f_5_5 m, f_5 v, int k, int n, int row, i_7 cols)
```

```
   {
   int    i;
   int    j;
   float  f;

   for (i = k+1; i <= n; i++)
      {
      f = -m[i][k] / m[k][k];
      for (j = k; j <= n; j++)
         {
         m[i][j] = m[i][j] + f * m[k][j];
         if ( j == k )
               lcprintf (row+i-1, cols[j], 0);
            else
               lcprintf (row+i-1, cols[j], m[i][j]);
         pause (500);
         }
      v[i] = v[i] + f * v[k];
      lcprintf (row+i-1, cols[7], v[i]);
      pause (500);
      }
   }

void solve_y_vector (f_5_5 m, f_5 v, int n, f_5 y, int row, i_7 cols)
   {
   int  i;
   int  j;

   for (i = n; i >= 1; i--)
      {
      for (j = i+1; j <= n; j++)
         {
         v[i] = v[i] - m[i][j] * y[j];
         lcprintf (row+i-1, cols[j], 0);
         lcprintf (row+i-1, cols[7], v[i]);
         pause (500);
         }
      y[i] = v[i] / m[i][i];
      lcprintf (row+i-1, cols[i], 1);
      lcprintf (row+i-1, cols[7], y[i]);
      pause (500);
      }
   }

void rev_sol_via_deriv (f_5 theta, f_3_2 noap, f_6 delta_trans_rot)
   {
   int    i;
   f_5    s;
   f_5    c;
   char   query_ch;
   f_3_2  dnoap;
   int    mr;
   i_3    mc;
   i_1    vr;
   i_1    vc;
   f_5    dtheta;

   sin_cos (theta, s, c);
lcputs (1, 0, "here "); cprintf ("%8.3f", s[3]);
   dsply_rsvd (&mr, mc, vr, vc);
   do
      {
/*      get_delta_trans_rot (delta_trans_rot, vr[0], vc[0]);*/
delta_trans_rot[1] = 196.548;
delta_trans_rot[2] =  64.286;
delta_trans_rot[3] = -75.924;
delta_trans_rot[4] =   0.107;
delta_trans_rot[5] =   0.341;
delta_trans_rot[6] =   1.085;
      calc_delta_noap (noap, delta_trans_rot, dnoap, mc, mr);
      calc_delta_theta (s, c, dnoap, noap, dtheta, vr[1], vc[1]);
      query_ch = cont ("different rates");
      }
      while ( query_ch == 'Y' );
   }

void dsply_rsvd (int *mr, i_3 mc, i_1 vr, i_1 vc)
   {
   int  lm;
   int  rm;

   lm = 16;
   rm = 15;
   locate (9, 0);
   mcputs (lm, rm, "        Reverse Solutions via Derivatives         ");
   mcputs (lm, rm, "  Delta Trans & Rots            Delta Thetas      ");
   mcputs (lm, rm, " ¦tx¦   ¦          ¦        ¦DT1¦   ¦           ¦ ");
   mcputs (lm, rm, " ¦ty¦   ¦          ¦        ¦DT2¦   ¦           ¦ ");
   mcputs (lm, rm, " ¦tz¦ = ¦          ¦        ¦DT3¦ = ¦           ¦ ");
   mcputs (lm, rm, " ¦rx¦   ¦          ¦        ¦DT4¦   ¦           ¦ ");
   mcputs (lm, rm, " ¦ry¦   ¦          ¦        ¦DT5¦   ¦           ¦ ");
   mcputs (lm, rm, " ¦rz¦   ¦          ¦                              ");
   mcputs (lm, rm, "                                                  ");
   mcputs (lm, rm, "          dN         dO         dA         dP     ");
   mcputs (lm, rm, "    ¦                                            ¦");
   mcputs (lm, rm, "dT: ¦                                            ¦");
   mcputs (lm, rm, "    ¦                                            ¦");
   mcputs (lm, rm, "    ¦     0          0          0          1     ¦");

   vr[0] = 11;
   vc[0] = 25;
   vr[1] = 11;
   vc[1] = 53;
   *mr   = 19;
   mc[0] = 21;
   mc[1] = 32;
   mc[2] = 43;
   mc[3] = 54;
   }

void get_delta_trans_rot (f_6 delta_trans_rot, int vr, int vc)
   {
   int    i;
   float  value;
   c_6_9  rate_lbl;

   strcpy(rate_lbl[1], "transl x");
   strcpy(rate_lbl[2], "transl y");
   strcpy(rate_lbl[3], "transl z");
   strcpy(rate_lbl[4], "rotate x");
   strcpy(rate_lbl[5], "rotate y");
   strcpy(rate_lbl[6], "rotate z");

   for (i = 1; i <= 6; i++)
      {
      lcputs (23, 20, "Enter d ");
      cputs (rate_lbl[i]);
      cputs (" :  ");
      lcputs (24, 22, "<Return> only to leave value unchanged as");
      lcprintf (24, 64, delta_trans_rot[i]);
      value = indec (23, 40);
      if ( value != 1000 )
         delta_trans_rot[i] = value;
      erase_prompt (23);
      lcprintf (vr+i-1, vc, delta_trans_rot[i]);
      }
   }

void calc_delta_noap (f_3_2 noap, f_6 dtr, f_3_2 dnoap,
                      i_3 mc, int mr)
   {
   int  i;
   int  j;

   for (i = 0; i <= 2; i++)
      {
      dnoap[i][0] = -noap[i][1]*dtr[6] + noap[i][2]*dtr[5];
      dnoap[i][1] =  noap[i][0]*dtr[6] - noap[i][2]*dtr[4];
      dnoap[i][2] = -noap[i][0]*dtr[5] + noap[i][1]*dtr[4];
      }

   for (i = 0; i <= 2; i++)
      dnoap[3][i] =  dtr[i+1];

   for (i = 0; i <= 3; i++)
      for (j = 0; j <= 2; j++)
         lcprintf (mr+j, mc[i], dnoap[i][j]);
   }

void calc_delta_theta (f_5 s, f_5 c, f_3_2 dnoap, f_3_2 noap,
                       f_5 dtheta, int vr, int vc)
   {
   int  div_zero;
   int  i;

   delta_theta312 (s, c, dnoap, noap, dtheta, &div_zero);
   if ( !div_zero )
         {
         delta_theta4 (s, c, dnoap, noap, dtheta);
         delta_theta5 (s, c, dnoap, noap, dtheta);
         for (i = 1; i <= 5; i++)
            lcprintf (vr+i-1, vc, dtheta[i]);
         }
      else
         {
         locate (22, 20);
         cprintf ("Zero Divide for Theta %d => No Solution", div_zero);
         }
   }

void delta_theta312 (f_5 s, f_5 c, f_3_2 dnoap, f_3_2 noap, f_5 dtheta,
                     int *div_zero)
   {
   int    i;
   f_2    arm;
   f_2    darm;
   float  denominator;
   float  factor;

   *div_zero = 0;

   for (i = 0; i <= 2; i++)
      {
       arm[i] =  noap[3][i] - d5* noap[2][i];
      darm[i] = dnoap[3][i] - d5*dnoap[2][i];
      }
lcputs (1, 0, "here 1");
   if ( fabs(s[3]) > tolerance )
         dtheta[3] = -(arm[0]*darm[0] +arm[1]*darm[1] +arm[2]*darm[2])
                    / (a2*a3*s[3]);
      else
         *div_zero = 3;
lcputs (2, 0, "here 2");
   if ( fabs(denominator = arm[0]*c[1] + arm[1]*s[1]) > tolerance )
         dtheta[1] = ( c[1]*darm[1] -s[1]*darm[0] - a3*c[3]*dtheta[3] )
                    / denominator;
      else
         *div_zero = 1;
lcputs (3, 0, "here 3");
   if ( fabs(factor = a3*c[3] + a2) > tolerance )
```

```
         if ( fabs(c[2]) > tolerance )
            dtheta[2] = (a3*s[2]*s[3]*dtheta[3] + darm[2])
                        / (c[2]*factor);
         else
            dtheta[2] = ( c[1]*darm[0] + s[1]*darm[1]
                          + (c[1]*arm[1] - s[1]*arm[0]) * dtheta[1]
                          + a3*c[2]*s[3]*dtheta[3] )
                        / (-s[2]*factor);
      else
         *div_zero = 2;
lcputs (4, 0, "here 4");
   }

void delta_theta4 (f_5 s, f_5 c, f_3_2 dnoap, f_3_2 noap, f_5 dtheta)
   {
   float  f1;
   float  f2;
   float  result;

   f1 = noap[2][0]*s[1] - noap[2][1]*c[1];
   f2 = noap[2][0]*c[1] + noap[2][1]*s[1];
   if ( fabs (s[4]) > tolerance )
         dtheta[4] = ( -c[1]*s[2]*dnoap[2][0]
                     + -s[1]*s[2]*dnoap[2][1]
                     +  c[2]*dnoap[2][2]
                     +  f1*s[2]*dtheta[1]
                     + -(f2*c[2] + dnoap[2][2]*s[2])*dtheta[2] ) / s[4];
      else
         dtheta[4] = ( (c[1]*c[2]*c[3] - s[1]*s[3])*dnoap[2][0]
                     + (s[1]*c[2]*c[3] + c[1]*s[3])*dnoap[2][1]
                     + s[2]*c[3]*dnoap[2][2]
                     + (-f1*c[2]*c[3] - f2*s[3])*dtheta[1]
                     + (-f2*s[2] + noap[2][2]*c[2])*c[3]*dtheta[2]
                     + ((-f2*c[2] - noap[2][2]*s[2])*s[3]
                       - f1*c[3])*dtheta[3] ) / c[4];
   }

void delta_theta5 (f_5 s, f_5 c, f_3_2 dnoap, f_3_2 noap, f_5 dtheta)
   {
   float  f1;
   float  f2;
   float  result;

   if ( fabs (s[5]) > tolerance )
         {
         f1 = -noap[1][0]*s[1] + noap[1][1]*c[1];
         f2 =  noap[1][0]*c[1] + noap[1][1]*s[1];
         dtheta[5] = ( (c[1]*c[2]*s[3] + s[1]*c[3])*dnoap[1][0]
                     + (s[1]*c[2]*s[3] - c[1]*c[3])*dnoap[1][1]
                     + s[2]*s[3]*dnoap[1][2]
                     + ( f1*c[2]*s[3] + f2*c[3])*dtheta[1]
                     + (-f2*s[2] + noap[1][2]*c[2])*s[3]*dtheta[2]
                     + ((f2*c[2] + noap[1][2]*s[2])*c[3] + f1*s[3])
                       * dtheta[3] )
                   / -s[5];
         }
      else
         {
         f1 = -noap[0][0]*s[1] + noap[0][1]*c[1];
         f2 =  noap[0][0]*c[1] + noap[0][1]*s[1];
         dtheta[5] = ( (c[1]*c[2]*s[3] + s[1]*c[3])*dnoap[0][0]
                     + (s[1]*c[2]*s[3] - c[1]*c[3])*dnoap[0][1]
                     +  s[2]*s[3]*dnoap[0][2]
                     + ( f1*c[2]*s[3] + f2*c[3])*dtheta[1]
                     + (-f2*s[2] + noap[0][2]*c[2])*s[3]*dtheta[2]
                     + ((f2*c[2] + noap[0][2]*s[2])*c[3] + f1*s[3])
                       * dtheta[3] )
                   / c[5];
         }
   }
```

APPENDIX G

TRAJECTORY CONTROL PROCEDURES LISTING

```
#include <c:\ed's\header.c>

void trajectory_control ( )
   {
   int      n;
   f_5_10   theta;
   f_9      t;
   f_5_9    a;
   f_5_9    b;
   f_5_9    c;
   f_5_9    d;
   f_5_9    e;
   f_5     theta_lb;
   f_5     theta_hb;
   f_5     theta_vmax;
   f_5     theta_amax;
   float   f;
   float   f_current;
   int     i;
   int     j;
   char    *prompt_msg1;
   char    *prompt_msg2;
   char    qc;

   dsply_trajectory_introduction ( );
   wait_then_erase (9);
   for (i = 1; i <= 5; i++)
      for (j = 1; j <= 10; j++)
         theta[i][j] = 0;
   do
      {
      n = nodes_and_distances (theta, t);
      f = 1000000;
      for (i = 1; i <= 5; i++)
         {
         lcputs (9, 29, "Trajectory for Joint ");
         cprintf ("%d", i);
         calc_polynomials (n, theta[i], t, a[i], b[i], c[i], d[i],
                           e[i]);
         theta_lb[i] = -360;
         theta_hb[i] =  360;
         critical_positions (n, a[i], b[i], c[i], d[i], e[i], t,
                             theta_lb[i], theta_hb[i]);
         theta_vmax[i] = 100;
         theta_amax[i] = 100;
         f_current = traj_scaling (n, b[i], c[i], d[i], e[i], t,
                                   theta_vmax[i], theta_amax[i]);
         if ( f_current < f )
            f = f_current;
         }
      determine_positions (a, b, c, d, e, n, t, f);

      prompt_msg1 = "Continue with a new trajectory determination? (y/n)";
      prompt_msg2 = "";
      qc = prompt_input_char (prompt_msg1, prompt_msg2);
      }
      while ( qc == 'Y' );
   wait_then_erase (8);
   }

void dsply_trajectory_introduction ( )
   {
   int  lm;
   int  rm;

   locate (8, 0);
   lm = 16;
   rm = 16;
   mcputs (lm, rm, "                    Trajectory Control                   ");
   mcputs (lm, rm, "                                                         ");
```

```
      mcputs (lm, rm, "  This section creates spline polynomials over a");
      mcputs (lm, rm, "a set of path nodes defined by the user.  The   ");
      mcputs (lm, rm, "polynomials created will provide for continuity ");
      mcputs (lm, rm, "in terms of position, velocity, and accelera-   ");
      mcputs (lm, rm, "tion.  The process takes place in the following ");
      mcputs (lm, rm, "steps:                                          ");
      mcputs (lm, rm, "  1) input of trajectory nodes                  ");
      mcputs (lm, rm, "  2) determination of node velocities           ");
      mcputs (lm, rm, "  3) polynomial coefficient derivation          ");
      mcputs (lm, rm, "  4) spline extrema tests                       ");
      mcputs (lm, rm, "  5) scaling with regards to extreme velocities ");
      mcputs (lm, rm, "     and accelerations                          ");
      mcputs (lm, rm, "  6) evaluation of polynomial position at       ");
      mcputs (lm, rm, "     selected times                             ");
      }

   int nodes_and_distances (f_5_10 theta, f_9 t)
      {
      int  row;
      i_6  cols;
      int  n;

      dsply_nodes_dists (&row, cols);
      n = input_nodes (theta, row, cols);
      calc_distance (n, theta, t, row, cols[6]);
      wait_then_erase (10);
      return (n);
      }

   void dsply_nodes_dists (int *row, i_6 cols)
      {
      locate (10, 0);
      mcputs (21, 20, "Input of Nodes Along Desired Trajectory");

      *row    = 11;
      cols[0] =  5;
      cols[1] = 10;
      cols[2] = 20;
      cols[3] = 30;
      cols[4] = 40;
      cols[5] = 50;
      cols[6] = 60;
      }

   int input_nodes (f_5_10 p, int row, i_6 cols)
      {
      int    i;
      int    j;
      int    n;
      float  value;

      p[1][1] =    0;
      p[2][1] =    0;
      p[3][1] =    0;
      p[4][1] =    0;
      p[5][1] =    0;
      p[1][2] =   10;
      p[2][2] =    6;
      p[3][2] =   20;
      p[4][2] =   -5;
      p[5][2] =  180;
      p[1][3] =   25;
      p[2][3] =   12;
      p[3][3] =   40;
      p[4][3] =   -7;
      p[5][3] =   45;
      p[1][4] =    0;
      p[2][4] =    0;
      p[3][4] =   75;
      p[4][4] =   22;
```

```
    p[5][4] =   45;
    p[1][5] =   20;
    p[2][5] =   -3;
    p[3][5] =   80;
    p[4][5] =   12;
    p[5][5] =  -90;
    p[1][6] =  -75;
    p[2][6] =   10;
    p[3][6] =   60;
    p[4][6] =   35;
    p[5][6] =   90;
    p[1][7] = -115;
    p[2][7] =   25;
    p[3][7] =   50;
    p[4][7] =   65;
    p[5][7] =  -35;
    n = 7;

    do
       {
       lcputs (23, 20, "Number of Nodes (4 to 10):");
       n = inint (23, 48);
       erase_prompt (23);
       }
       while ( (n < 4) ¦ (n > 10) );

    for (i = 1; i <= n; i++)
       {
       locate (row+i-1, cols[0]);
       cprintf ("%2d", i);
       for (j = 1; j <= 5; j++)
          {
          locate (23, 20);
          cprintf ("Enter Theta for node %d, joint %d:", i, j);
          lcputs (24, 22, "<Return> only to leave unchanged as");
          lcprintf (24, 58, p[j][i]);
          value = indec (23, 53);
          if ( value != 1000 )
             p[j][i] = value;
          lcprintf (row+i-1, cols[j], p[j][i]);
          erase_prompt (23);
          }
       }
    return (n);
    }

void calc_distance (int n, f_5_10 p, f_9 t, int row, int col)
    {
    int  i;
    int  j;

    for (i = 1; i <= n-1; i++)
       {
       t[i] = 0;
       for (j = 1; j <= 5; j++)
          t[i] += square (p[j][i+1] - p[j][i]);
       t[i] = sqrt(t[i]);
       lcprintf (row+i-1, col, t[i]);
       }
    }

void calc_polynomials (int n, f_10 p, f_9 t,
                       f_9 a, f_9 b, f_9 c, f_9 d, f_9 e)
    {
    f_9  vel;

    calc_node_velocities (n, p, t, vel);
    calc_coefficients (n, p, t, vel, a, b, c, d, e);
    }
```

```
void calc_node_velocities (int n, f_10 p, f_9 t, f_9 vel)
   {
   int row;
   i_4 cols;
   f_9_2 coeff;
   f_9 rhs;

   dsply_node_velocities (&row, cols);
   equate_quartic_cubic_accs (t, p, coeff, rhs, row, cols);
   equate_cubic_accs (n, t, p, coeff, rhs, row, cols);
   equate_cubic_quartic_accs (n, t, p, coeff, rhs, row, cols);
   wait_then_continue ( );
   forward_eliminate_term1 (n, coeff, rhs, row, cols);
   wait_then_continue ( );
   backward_eliminate_term3 (n, coeff, rhs, vel, row, cols);
   wait_then_erase (10);
   }

void dsply_node_velocities (int *row, i_4 cols)
   {
   locate (10, 0);
   mcputs (33, 32, "Node Velocities");
   mcputs (16, 15, " i   vel(i-1)      vel(i)    vel(i+1)          rhs");

   *row = 12;
   cols[0] = 17;
   cols[1] = 19;
   cols[2] = 31;
   cols[3] = 43;
   cols[4] = 55;
   }

void equate_quartic_cubic_accs (f_9 t, f_10 p, f_9_2 coeff, f_9 rhs,
                                int row, i_4 cols)
   {
   lcputs (row, cols[0], "2");

   coeff[2][1] = 2*t[1] + 3*t[2];
   lcprintf (row, cols[2], coeff[2][1]);

   coeff[2][2] = t[1];
   lcprintf (row, cols[3], coeff[2][2]);

   rhs[2] = (6 * t[2]/t[1]) * (p[2] - p[1]) +
            (3 * t[1]/t[2]) * (p[3] - p[2]);
   lcprintf (row, cols[4], rhs[2]);
   }

void equate_cubic_accs (int n, f_9 t, f_10 p, f_9_2 coeff, f_9 rhs,
                        int row, i_4 cols)
   {
   int  i;

   for (i = 3; i <= n-2; i++)
      {
      locate (row+i-2, cols[0]);
      cprintf ("%d", i);

      coeff[i][0] = t[i];
      lcprintf (row+i-2, cols[1], coeff[i][0]);

      coeff[i][1] = 2 * (t[i] + t[i-1]);
      lcprintf (row+i-2, cols[2], coeff[i][1]);

      coeff[i][2] = t[i-1];
      lcprintf (row+i-2, cols[3], coeff[i][2]);

      rhs[i] = 3 * (t[i]/t[i-1]) * (p[i] - p[i-1]) +
               3 * (t[i-1]/t[i]) * (p[i+1] - p[i]);
      lcprintf (row+i-2, cols[4], rhs[i]);
      }
   }

void equate_cubic_quartic_accs (int n, f_9 t, f_10 p, f_9_2 coeff,
                                f_9 rhs, int row, i_4 cols)
   {
   locate (row+n-3, cols[0]);
   cprintf ("%d", n-1);

   coeff[n-1][0] = t[n-1];
   lcprintf (row+n-3, cols[1], coeff[n-1][0]);

   coeff[n-1][1] = 2*t[n-1] + 3*t[n-2];
   lcprintf (row+n-3, cols[2], coeff[n-1][1]);

   rhs[n-1] = (3 * t[n-1] / t[n-2]) * (p[n-1] - p[n-2])
            + (6 * t[n-2] / t[n-1]) * (p[n]   - p[n-1]);
   lcprintf (row+n-3, cols[4], rhs[n-1]);
   }

void forward_eliminate_term1 (int n, f_9_2 coeff, f_9 rhs,
                              int row, i_4 cols)
   {
   int    i;
   float  multiplier;

   for (i = 3; i <= n-1; i++)
      {
      multiplier = -coeff[i][0] / coeff[i-1][1];
      lcputs (row+i-2, cols[1], "           ");
      pause (short_pause);

      coeff[i][1] = coeff[i][1] + multiplier * coeff[i-1][2];
      lcprintf (row+i-2, cols[2], coeff[i][1]);
      pause (short_pause);

      rhs[i] = rhs[i] + multiplier * rhs[i-1];
      lcprintf (row+i-2, cols[4], rhs[i]);
      pause (short_pause);
      }
   }

void backward_eliminate_term3 (int n, f_9_2 coeff, f_9 rhs, f_9 vel,
                               int row, i_4 cols)
   {
   int  i;

   lcprintf (row+n-3, cols[2], 1);

   vel[n-1] = rhs[n-1] / coeff[n-1][1];
   lcprintf (row+n-3, cols[4], vel[n-1]);

   for (i = n-2; i >= 2; i--)
      {
      lcputs (row+i-2, cols[3], "           ");

      rhs[i] = rhs[i] - coeff[i][2] * vel[i+1];
      lcprintf (row+i-2, cols[4], rhs[i]);
      pause (short_pause);

      lcprintf (row+i-2, cols[2], 1);

      vel[i] = rhs[i] / coeff[i][1];
      lcprintf (row+i-2, cols[4], vel[i]);
      pause (short_pause);
      }
   }

void calc_coefficients (int n, f_10 p, f_9 t, f_9 vel,
                        f_9 a, f_9 b, f_9 c, f_9 d, f_9 e)
   {
   int  row;
   i_5  cols;

   dsply_coefficients (&row, cols);
   calc_starting_quartic       (p, t, vel, a, b, c, d, e, row, cols);
   calc_intermediate_cubics (n, p, t, vel, a, b, c, d, e, row, cols);
   calc_ending_quartic      (n, p, t, vel, a, b, c, d, e, row, cols);
   wait_then_erase (10);
   }

void dsply_coefficients (int *row, i_5 cols)
   {
   locate (10, 0);
   mcputs (29, 28, "Polynomial Coefficients");
   mcputs (18, 13, "A           B           C           D           E");
   mcputs (18, 13, "                                                 ");
   *row    = 12;
   cols[0] =  9;
   cols[1] = 13;
   cols[2] = 25;
   cols[3] = 37;
   cols[4] = 49;
   cols[5] = 61;
   }

void calc_starting_quartic (f_10 p, f_9 t, f_9 vel,
                            f_9 a, f_9 b, f_9 c, f_9 d, f_9 e,
                            int row, i_5 cols)
   {
   lcputs (row, cols[0], " 1");

   a[1] = p[1];
   lcprintf (row, cols[1], a[1]);

   b[1] = 0;
   lcprintf (row, cols[2], b[1]);

   c[1] = 0;
   lcprintf (row, cols[3], c[1]);

   d[1] = ( 4 / pow(t[1],3)) * (p[2] - p[1])
        - ( 1 / square (t[1])) * vel[2];
   lcprintf (row, cols[4], d[1]);

   e[1] = (-3 / pow(t[1],4)) * (p[2] - p[1])
        + ( 1 / pow(t[1],3)) * vel[2];
   lcprintf (row, cols[5], e[1]);
   }

void calc_intermediate_cubics (int n, f_10 p, f_9 t, f_9 vel,
                               f_9 a, f_9 b, f_9 c, f_9 d, f_9 e,
                               int row, i_5 cols)
   {
   int  i;

   for (i = 2; i <= n-2; i++)
      {
      locate (row+i-1, cols[0]);
      cprintf ("%2d", i);

      a[i] = p[i];
      lcprintf (row+i-1, cols[1], a[i]);

      b[i] = vel[i];
      lcprintf (row+i-1, cols[2], b[i]);

      c[i] = (3/square (t[i])) * (p[i+1] - p[i])
           - (1/t[i]) * (vel[i+1] + 2*vel[i]);
      lcprintf (row+i-1, cols[3], c[i]);
      d[i] = -(2/pow(t[i],3)) * (p[i+1] - p[i])
           +  (1/square (t[i])) * (vel[i+1] + vel[i]);
      lcprintf (row+i-1, cols[4], d[i]);

      e[i] = 0;
      }
   }

void calc_ending_quartic (int n, f_10 p, f_9 t, f_9 vel,
                          f_9 a, f_9 b, f_9 c, f_9 d, f_9 e,
                          int row, i_5 cols)
   {
   locate (row+n-2, cols[0]);
   cprintf ("%2d", n-1);

   a[n-1] = p[n-1];
   lcprintf (row+n-2, cols[1], a[n-1]);

   b[n-1] = vel[n-1];
   lcprintf (row+n-2, cols[2], b[n-1]);

   c[n-1] =  (6 / square (t[n-1])) * (p[n] - p[n-1])
          -  (3 / t[n-1]) * vel[n-1];
   lcprintf (row+n-2, cols[3], c[n-1]);

   d[n-1] = -(8 / pow(t[n-1],3)) * (p[n] - p[n-1])
          +  (3 / square (t[n-1])) * vel[n-1];
   lcprintf (row+n-2, cols[4], d[n-1]);

   e[n-1] =  (3 / pow(t[n-1],4)) * (p[n] - p[n-1])
          -  (1 / pow(t[n-1],3)) * vel[n-1];
   lcprintf (row+n-2, cols[5], e[n-1]);
   }

void critical_positions (int n, f_9 a, f_9 b, f_9 c, f_9 d, f_9 e,
                         f_9 t, int lb, int hb)
   {
   int  row;
   i_7  cols;
   int  i;

   dsply_crit_pos (&row, cols);
   terminal_crit_pos (1, a[1], b[1], c[1], d[1], e[1], t[1], lb, hb,
                      row, cols);

   for (i = 2; i <= n-2; i++)
      intermediate_crit_pos (i, a[i], b[i], c[i], d[i], t[i], lb, hb,
                             row+i-1, cols);

   terminal_crit_pos (n-1, a[n-1], b[n-1], c[n-1], d[n-1], e[n-1],
                      t[n-1], lb, hb, row+n-2, cols);
   wait_then_erase (10);
   }

void dsply_crit_pos (int *row, i_7 cols)
   {
   locate (10, 0);
   mcputs (31, 31, "Critical Positions");
   cputs (" i       time    position        time    position    ");
   cputs ("    time    position    bounds");

   *row    = 12;
   cols[0] =  1;
   cols[1] =  4;
   cols[2] = 15;
   cols[3] = 27;
   cols[4] = 38;
   cols[5] = 50;
   cols[6] = 61;
   cols[7] = 75;
   }

void terminal_crit_pos (int i, float a, float b, float c, float d,
                        float e, float t, float lb, float hb,
                        int row, i_7 cols)
   {
   float  q;
   float  r;
   float  discr;
   int    roots;
   f_2    tcp;
   f_2    cp;
   float  x;
   int    j;

   locate (row, cols[0]);
   cprintf ("%d", i);
   q = (1 / (2 * e)) * ((c / 3) - square (d) / (8 * e));
   r = (1 / (8 * e)) * ((d / (2*e)) * (c - square (d) / (4 * e)) - b);
   discr = pow (q,3) + square (r);

   if ( discr > small_tolerance )
         {
         roots = 1;
         tcp[0] = cube_root (r + sqrt(discr))
                + cube_root (r - sqrt(discr)) - d / (4 * e);
         }
      else
         if ( fabs(discr) < small_tolerance )
               {
               roots = 2;
               tcp[0] = 2*cube_root (r) - d / (4 * e);
               tcp[1] =  -cube_root (r) - d / (4 * e);
               }
            else
               {
               roots = 3;
               x = atan2(sqrt(-discr), r);
               tcp[0] = 2 * sqrt(-q) * cos(x/3) - d/(4*e);
               tcp[1] = -sqrt(-q) * ( cos(x/3) + sqrt(3)*sin(x/3))
                      - d/(4*e);
               tcp[2] =  sqrt(-q) * (-cos(x/3) + sqrt(3)*sin(x/3))
                      - d/(4*e);
               }
   for (j = 0; j <= roots-1; j++)
      cp[j] = eval_cp (a, b, c, d, e, tcp[j], t,
                       row, cols[j*2+1], cols[j*2+2]);
   for (j = 0; j <= roots-1; j++)
      check_range (lb, hb, cp[j], row, cols[7]);
   }

void intermediate_crit_pos (int i, float a, float b, float c, float d,
                            float t, float lb, float hb,
                            int row, i_7 cols)
   {
   int    j;
   float  discr;
   int    roots;
   f_1    tcp;
   f_1    cp;

   locate (row, cols[0]);
   cprintf ("%d", i);
   discr = square (c) - 3*b*d;
   if (discr > small_tolerance)
         {
         roots = 2;
         tcp[0] = (-c + sqrt(discr)) / (3 * d);
         tcp[1] = (-c - sqrt(discr)) / (3 * d);
```

```
          }
       else
          if ( fabs(discr) < small_tolerance )
                {
                roots = 1;
                tcp[0] = -c / (3 * d);
                }
             else
                roots = 0;
   for (j = 0; j <= roots-1; j++)
      cp[j] = eval_cp (a, b, c, d, 0, tcp[j], t,
                          row, cols[j*2+1], cols[j*2+2]);
   for (j = 0; j <= roots-1; j++)
      check_range (lb, hb, cp[j], row, cols[7]);
   }

float eval_cp (float a, float b, float c, float d, float e,
               float tcp, float t, int row, int tcol, int pcol)
   {
   float  cp;

   lcprintf (row, tcol, tcp);
   if ( (tcp >= 0) & (tcp <= t) )
         {
         cp = a + b*tcp + c*square (tcp) + d*pow(tcp,3) + e*pow(tcp,4);
         lcprintf (row, pcol, cp);
         }
      else
         {
         cp = 0;
         lcputs (row, pcol, "  time out");
         }
   return (cp);
   }

void check_range (float lb, float hb, float cp, int row, int col)
   {
   if ( (cp >= lb) & (cp <= hb) )
         lcputs (row, col, " In");
      else
         lcputs (row, col, "Out");
   }

float traj_scaling (int n, f_9 b, f_9 c, f_9 d, f_9 e, f_9 t,
                          float theta_maxv, float theta_maxa)
   {
   int     row;
   i_7     cols;
   float   maxv;
   float   maxa;
   int     i;
   float   cv;
   float   ca;
   float   fv;
   float   fa;
   float   f;

   dsply_traj_scaling (&row, cols);
   maxv = terminal_crit_vel (1, b[1], c[1], d[1], e[1], t[1],
                                  row, cols);
   maxa = terminal_crit_acc (c[1], d[1], e[1], t[1], row, cols);
   for (i = 2; i <= n-2; i++)
      {
      cv = intermediate_crit_vel (i, b[i], c[i], d[i], t[i],
                                       row+i-1, cols);
      if ( cv > maxv )
         maxv = cv;
      ca = intermediate_crit_acc (c[i], row+i-1, cols);
      if ( ca > maxa )
         maxa = ca;
      }
   cv = terminal_crit_vel (n-1, b[n-1], c[n-1], d[n-1], e[n-1], t[n-1],
                           row+n-2, cols);
   if ( cv > maxv )
      maxv = cv;
   ca = terminal_crit_acc (c[n-1], d[n-1], e[n-1], t[n-1],
                           row+n-2, cols);
   if ( ca > maxa )
      maxa = ca;

   lcprintf (row+ 9, cols[7], maxv);
   lcprintf (row+10, cols[7], maxa);

   fv = theta_maxv / maxv;
   fa = sqrt(theta_maxa / maxa);
   if ( fv < fa )
         f = fv;
      else
         f = fa;
   lcprintf (row+11, cols[7], f);
   wait_then_erase (9);
   return (f);
   }

void dsply_traj_scaling (int *row, i_7 cols)
   {
   locate (10,0);
   cputs ("                        Critical Velocities              ");
   cputs ("   Critical Acceleration    ");
   cputs ("    i       time     velocity       time     velocity");
   cputs ("       time  acceleration  ");
   locate (21, 0);
   mcputs (16, 16, "Maximum Velocity:                                ");
   mcputs (16, 16, "Maximum Acceleration:                            ");
   mcputs (16, 16, "Scaling Factor:                                  ");

   *row    = 12;
   cols[0] =  3;
   cols[1] =  8;
   cols[2] = 19;
   cols[3] = 32;
   cols[4] = 43;
   cols[5] = 56;
   cols[6] = 67;
   cols[7] = 38;
   }

float terminal_crit_vel (int i, float b, float c, float d, float e,
                         float t, int row, i_6 cols)
   {
   int     j;
   float   discr;
   int     roots;
   f_1     tcv;
   float   maxv;
   f_1     cv;
   float   mag;

   locate (row, cols[0]);
   cprintf ("%2d", i);
   discr = square (d) - 8*c*e/3;
   if ( discr > small_tolerance)
         {
         roots = 2;
         tcv[0] = (-d + sqrt(discr)) / (4*e);
         tcv[1] = (-d - sqrt(discr)) / (4*e);
         }
      else
         if ( fabs(discr) < small_tolerance )
               {
                roots = 1;
                tcv[0] = -d / (4*e);
                }
             else
                roots = 0;
    for (j = 0; j <= roots-1; j++)
       cv[j] = eval_cv (b, c, d, e, tcv[j], t,
                         row, cols[2*j+1], cols[2*j+2]);
    maxv = 0;
    for (j = 0; j <= roots-1; j++)
       if ( (mag = fabs(cv[j])) > maxv )
          maxv = mag;
    return (maxv);
    }

float intermediate_crit_vel (int i, float b, float c, float d, float t,
                                int row, i_6 cols)
   {
   float  tcv;
   float  cv;

   locate (row, cols[0]);
   cprintf ("%2d  ", i);
   tcv = -c / (3*d);
   cv = eval_cv (b, c, d, 0, tcv, t, row, cols[1], cols[2]);
   return (fabs(cv));
   }

float eval_cv (float b, float c, float d, float e, float tcv, float t,
               int row, int tcol, int vcol)
   {
   float  cv;

   lcprintf (row, tcol, tcv);
   if ( (tcv >= 0) & (tcv <= t) )
         {
         cv = b + 2*c*tcv + 3*d*square (tcv) + 4*e*pow(tcv, 3);
         lcprintf (row, vcol, cv);
         }
      else
         {
         cv = 0;
         lcputs (row, vcol, "  time out");
         }
   return (cv);
   }

float terminal_crit_acc (float c, float d, float e, float t,
                           int row, i_6 cols)
   {
   float  tca;
   float  ca;

   tca = -d / (4*e);
   lcprintf (row, cols[5], tca);
   if ( (tca >= 0) & (tca <= t) )
         {
         ca = 2*c - (3*square (d)) / (4*e);
         lcprintf (row, cols[6], ca);
         }
      else
         {
         ca = 0;
         lcputs (row, cols[6], " time out ");
         }
   return (fabs(ca));
   }

float intermediate_crit_acc (float c, int row, i_6 cols)
   {
```

```
   float  ca;

   ca = 2*c;
   lcprintf (row, cols[5], 0);
   lcprintf (row, cols[6], ca);
   return (fabs(ca));
   }

void determine_positions (f_5_9 a, f_5_9 b, f_5_9 c, f_5_9 d, f_5_9 e,
                          int n, f_9 t, float f)
   {
   i_2    rows;
   i_4    time_cols;
   i_6    eval_cols;
   int    scale_col;
   f_9    cumulative_time;
   float  total_time;
   int    i;
   float  trajectory_pos;
   float  real_time;
   float  scale_time;
   int    j;

   dsply_det_positions (rows, time_cols, eval_cols, &scale_col);
   lcprintf (rows[2], scale_col, f);
   cumulative_time[0] = 0;
   for (i = 1; i <= n-1; i++)
      {
      locate (rows[0]+i-1, time_cols[0]);
      cprintf ("%d", i);
      lcprintf (rows[0]+i-1, time_cols[1], 0);
      lcprintf (rows[0]+i-1, time_cols[2], t[i]);

      cumulative_time[i] = cumulative_time[i-1] + t[i]/f;
      lcprintf (rows[0]+i-1, time_cols[3], cumulative_time[i-1]);
      lcprintf (rows[0]+i-1, time_cols[4], cumulative_time[i]);
      }

   while ( (real_time = get_time (cumulative_time, n)) != -1 )
      {
      i = 0;
      do
         i++;
         while ( real_time > cumulative_time[i] );
      locate (rows[1], eval_cols[0]);
      cprintf ("%d", i);
      scale_time = (real_time - cumulative_time[i-1]) * f;
      lcprintf (rows[1], eval_cols[6], scale_time);
      for (j = 1; j <= 5; j++)
         {
         trajectory_pos = a[j][i] + b[j][i] * scale_time
                        + c[j][i] * square (scale_time)
                        + d[j][i] * pow (scale_time, 3)
                        + e[j][i] * pow (scale_time, 4);
         lcprintf (rows[1], eval_cols[j], trajectory_pos);
         }
      }
   wait_then_erase (9);
   }

void dsply_det_positions (i_2 rows, i_4 time_cols, i_6 eval_cols,
                          int *scale_col)
   {
   locate (9, 0);
   cputs ("  Determination of Trajectory Position at Arbi");
   cputs ("trary Times  (scale =          ) ");
   cputs ("           Segment       Scale Time Range     ");
   cputs ("      Real Time Range                ");
   cputs ("                                             ");
   cputs ("                                     ");
   locate (20, 0);
   cputs ("Segment   Scale Time     Theta 1     Theta 2");
   cputs ("     Theta 3     Theta 4     Theta 5");
   cputs ("                                            ");
   cputs ("                                    ");

   rows[0] = 11;
   time_cols[0] = 14;
   time_cols[1] = 20;
   time_cols[2] = 33;
   time_cols[3] = 46;
   time_cols[4] = 59;

   rows[1] = 21;
   eval_cols[0] =  3;
   eval_cols[6] = 10;
   eval_cols[1] = 22;
   eval_cols[2] = 34;
   eval_cols[3] = 46;
   eval_cols[4] = 58;
   eval_cols[5] = 70;

   rows[2] = 9;
   *scale_col = 68;
   }

float get_time (f_9 cumulative_time, int n)
   {
   float   real_time;

   lcputs (23, 20, "Enter cumulative real time for evaluation:  ");
   lcputs (24, 20, "<-1> to terminate");
   do
      {
      lcputs (23,20+44, "          ");
      real_time = indec (23, 20+44);
      }
      while ( ((real_time < 0) ¦ (real_time > cumulative_time[n-1]))
            & (real_time != -1) );
   erase_prompt (23);
   return (real_time);
   }
```

BIBLIOGRAPHY

[Bana85] Banas, J. "Computer-Controlled Robot Arm," *Radio Electronics*, May 1985, pp. 49-53, 117.

[Bedf64] Bedford, B. D. and Haft, R. G. *Principles of Inverter Circuits*. New York: John Wiley and Sons, Inc., 1964.

[Boul71] Boullion, T. L. and Odell, P. L. *Generalized Inverse Matrices*. New York: Wiley-Interscience, 1971.

[Brad83] Brady, M., et. al., eds. *Robot Motion: Planning and Control*. Cambridge, MA: The MIT Press, 1983.

[Card85] Cardoza, A. and Vlk, S. J. *Robotics*. Blue Ridge Summit, PA: Tab Books, Inc., 1985.

[Casa73] Casasent, D. *Electronic Circuits*. New York: Quantum Publishers, Inc., 1973.

[Coif83a] Coiffet, P. *Robot Technology: Modelling and Control*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1983.

[Coif83b] Coiffet, P. and Chirouze, M. *An Introduction to Robot Technology*. New York: McGraw-Hill Book Company, 1983.

[Cook82] Cook, C. C. and Ho, C. Y. "The Application of Spline Functions to Trajectory Generation for Computer-Controlled Manipulators," *Digital Systems for Industrial Automation*, Vol. 1, No. 4, 1982, pp. 325-333.

[Dena55] Denavit, J. and Hartenberg, R. S. "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," *Journal of Applied Mechanics*, June 1955, pp. 215-221.

[Desa85] Desa, S. and Roth, B. "Mechanics: Kinematics and Dynamics." In *Recent Advances in Robotics*. Beni, G. and Hackwood, S., eds. New York: Wiley-Intersciences, 1985.

[Duff80] Duffy, J. *Analysis of Mechanisms and Robot Manipulators*. New York: Halstead Press, 1980.

[Feat83] Featherstone, R. "Position and Velocity Transformations Between Robot End-Effector Coordinates and Joint Angles," *The International Journal of Robotics Research*, Vol. 2, No. 2, Summer 1983, pp. 35-45.

[Fole83] Foley, J. D. and Van Dam, A. *Fundamentals of Interactive Computer Graphics*. Reading, MA: Addison-Wesley Publishing Company, 1983.

[Gold85] Goldenberg, A. A., Benhabib, B., and Fenton, R. G. "A Complete Generalized Solution to the Inverse Kinematics of Robots," *IEEE Journal of Robotics and Automation*, Vol. RA-1, No. 1, March 1985, pp. 14-20.

[Gros78] Grossman, D. D. and Taylor, R. H. "Interactive Generation of Object Models with a Manipulator," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-8, No. 9, September 1978, pp. 667-679.

[Hibb65] Hibberd, R. G. *Transistors: Principles and Applications*. New York: Hart Publishing Company, Inc., 1965.

[Ho82] Ho, C. Y. and Copeland, K. W. "Solution of Kinematic Equations for Robot Manipulators," *Digital Systems for Industrial Automation*, Vol. 1, No. 4, 1982, pp. 335-352.

[Ho86] Ho, C. Y. and Sriwattanathamma, J. "Differential Relationship of Kinematic Model and Speed Control Strategies for a Computer-controlled Robot Manipulator," *Robitica,* Vol. 4, 1986, pp. 155-161.

[Ho90] Ho, C. Y. and Sriwattanathamma, J. *Robot Kinematics.* Norwood, New Jersey: Ablex Publishing Corporation, 1990.

[Holl80] Hollerbach, J. M. "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-10, No. 11, November 1980, pp. 730-736.

[Holz86] Holzbock, W. G. *Robotic Technology: Principles and Practice.* New York: Van Nostrand Reinhold Company, 1986.

[Khal84] Khalil, W. "Trajectories Calculations in the Joint Space of Robots." In *Advanced Software in Robotics.* Danthine, A. and Geradin, M., eds. New York: Elsevier Science Publishers B. V., 1984.

[Kirc85] Kircanski, M. and Vukobratovic, M. "Trajectory Planning for Redundant Manipulators in the Presence of Obstacles." In *Theory and Practice of Robots and Manipulators - Proceedings of RoManSy '84: The Fifth CISM-IFToMM Symposium.* Morecki, A., Bianchi, G. and Kedzior, K., eds. Cambridge, MA: The MIT Press, 1985.

[Krug54] Krugman, L. M. *Fundamentals of Transistors.* New York: John F. Rider Publisher, Inc., 1954.

[Lee82] Lee, C. S. G. "Robot Arm Kinematics, Dynamics, and Control," *Computer,* December 1982, pp. 62-80.

[Lewi83] Lewin, M. H. *Logic Design and Computer Organization.* Reading, MA: Addison-Wesley Publishing Company, 1983.

[Malm69] Malmstadt, H. V. and Enke, C. G. *Digital Electronics for Scientists.* Menlo Park, CA: W. A. Benjamin, Inc., 1969.

[Nico85] Nicolaisen, P. "Occupational Safety and Industrial Robots - Present Stage of Discussions within the Tripartitie Group." In *Robot Technology and Applications.* Rathmill, K., et. al., eds. Berlin: Springer-Verlag, 1985.

[Nort88] Norton, P. and Wilton, R. *Programmer's Guide to the IBM PC & PS/2.* Redmond, WA: Microsoft Press, 1988.

[Pare85] Parent, M. and Laurgeau, C. *Robot Technology: Logic and Programming.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1985.

[Paul79] Paul, R. "Manipulator Cartesian Path Control," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-9, No. 11, November 1979, pp. 702-711.

[Paul81a] Paul, R. P., Shimano, B., and Mayer, G. E. "Kinematic Control Equations for Simple Manipulators," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-11, No. 6, June 1981, pp. 449-455.

[Paul81b] Paul, R. P., Shimano, B., and Mayer, G. E. "Differential Kinematic Control Equations for Simple Manipulators," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-11, No. 6, June 1981, pp. 456-460.

[Paul81c] Paul, R. P. *Robot Manipulators: Mathematics, Programming, and Control.* Cambridge, MA: The MIT Press, 1981.

[Paul87] Paul, R. P. and Zhang, H. "Computationally Efficient Kinematics for Manipulators with Spherical Wrists Based on the Homogeneous Transformation Representation." In *The Kinematics of Robot Manipulators.* McCarthy, J. M., ed. Cambridge, MA: The MIT Press, 1987.

[Rank85] Ranky, P. G. and Ho, C. Y. *Robot Modelling: Control and Applications with Software.* Bedford, England: IFS Publications Ltd., 1985.

[Rao71] Rao, C. R. *Generalized Inverse of Matrices and Its Applications.* New York: John Wiley and Sons, Inc., 1971.

[Robi84] Robillard, M. J. *Advanced Robot Systems.* Indianapolis, IN: Howard W. Sams and Co., Inc., 1984.

[Saff79] Safford, E. L. *The Complete Handbook of Robotics.* Blue Ridge Summit, PA: Tab Books, 1979.

[Spon89] Spong, M. W. and Vidyasagar, M. *Robot Dynamics and Control.* New York: John Wiley and Sons, Inc., 1989.

[Stra80] Strang, G. *Linear Algebra and Its Applications.* New York: Academic Press, 1980.

[Taka81] Takase, K., Paul, R. P., and Berg, E. J. "A Structured Approach to Robot Programming and Teaching," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-11, No. 4, April 1981, pp. 274-289.

[Vuko82] Vukobratovic, M. and Potkonjak, V. *Dynamics of Manipulation Robots: Theory and Application.* Berlin: Springer-Verlag, 1982.

[Walk66] Walker, R. L. *Introduction to Transistor Electronics.* Belmont, CA: Wadsworth Publishing Company, Inc., 1966.

[Zeld84] Zeldman, M. I. *What Every Engineer Should Know About Robots.* New York: Marcel Dekker, Inc., 1984.