1994-09

Optimal control of a two wheeled mobile robot

Emond, Bryan R.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

OPTIMAL CONTROL
OF A
TWO WHEELED MOBILE ROBOT

by

Bryan R. Emond

September, 1994

Thesis Advisor: R. Mukherjee

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 22, 1994.	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE OPTIMAL CONTROL OF A TWO WHEELED MOBILE ROBOT(U)		5. FUNDING NUMBERS
6. AUTHOR(S) Bryan R. Emond		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE *A

13. ABSTRACT *(maximum 200 words)*

Feedback control of a two wheeled mobile robot from one point in its configuration space to another presents a challenging problem. The mobile robot belongs to a class of systems with non-integrable motion constraints for which smooth feedback control laws cannot be designed. Recent work has been aimed at developing time-varying feedback control laws and piecewise smooth feedback control laws. These control techniques are, however, not optimal in any sense. In this research, we look into the optimal control of a mobile robot using partial feedback. A solution is obtained by application of Pontryagin's Minimization Principle and solving the associated two point boundary value problem using a numerical relaxation technique. The resulting robot trajectories exhibit optimal behavior for all non-trivial cases.

14. SUBJECT TERMS OPTIMAL, CONTROL, MOBILE ROBOT, NON-HOLONOMIC, PONTRYAGIN, LYAPUNOV, BROCKETT			15. NUMBER OF PAGES 166
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

OPTIMAL CONTROL
OF A
TWO WHEELED MOBILE ROBOT

by

Bryan R. Emond
Lieutenant, United States Coast Guard
B.S., U.S. Coast Guard Academy, 1985

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
SEPTEMBER 1994

Author: [redacted]
Bryan R. Emond

Approved by: [redacted]
Ranjan Mukherjee, Thesis Advisor

[redacted]
Matthew D. Kelleher, Chairman
Department of Naval/Mechanical Engineering

ABSTRACT

Feedback control of a two wheeled mobile robot from one point in its configuration space to another presents a challenging problem. The mobile robot belongs to a class of systems with non-integrable motion constraints for which smooth feedback control laws cannot be designed. Recent work has been aimed at developing time-varying feedback control laws and piecewise smooth feedback control laws. These control techniques are, however, not optimal in any sense. In this research, we look into the optimal control of a mobile robot using partial feedback. A solution is obtained by application of Pontryagin's Minimization Priciple and solving the associated two point boundary value problem using a numerical relaxation technique. The resulting robot trajectories exhibit optimal behavior for all non-trivial cases.

Thesis E4564 c.2

TABLE OF CONTENTS

I. INTRODUCTION

The mobile robot belongs to a class of systems with non-integrable motion constraints for which smooth feedback control laws for motion from one point in the configuration space to another cannot be designed [Ref. 1]. Recent work aims at developing time-varying feedback control laws [Ref. 1] and piecewise smooth feedback control laws [Ref. 2]. These control techniques are, however, not optimal in any sense. In this research, we look into the optimal control of a mobile robot using partial feedback.

A. KINEMATICS OF A MOBILE ROBOT

The position and orientation of a two wheeled mobile robot on a horizontal plane is described by three generalized coordinates. Figure 1 shows the three coordinates chosen for our robot. These are the two X-Y coordinates for the location of the robot on the plane, and an angular displacement, θ, to describe the robot's orientation with respect to the positive X axis.

The velocity of the robot can be described completely in terms of translation and rotation. Assuming no slipping, the interaction of the wheels with the plane restricts the instantaneous motion to the direction of orientation of the robot. Defining U_1 as the velocity in the direction of orientation, and U_2 as the rate of change of the orientation, the following constraint equations result:

$$\dot{X} = U_1 \cos\theta \tag{1}$$

$$\dot{Y} = U_1 \sin\theta \tag{2}$$

$$\dot{\theta} = U_2 \tag{3}$$

Note that while the constraints above limit the number of degrees of freedom for the system to two, specifically U_1 and U_2, a minimum of three coordinates are required to describe the system. This is true of all nonholonomic systems; the number of generalized coordinates required to describe the system is greater than the number of degrees of freedom.

A nonholonomic system is characterized by the non-integrable nature of the constraint between the first derivatives of the coordinates. [Ref. 4:p. 244] In the particular case of the mobile robot, the non-integrable constraint is due to the nature of the angular displacement term, θ. As θ is an independent function of time, the relationship between the remaining coordinates cannot be uniquely determined. In other words, for a robot moving from one position and orientation on the X-Y plane to another, the instantaneous value of θ depends upon the path followed by the robot. As a result, the coordinate relationship is dependent upon the path taken.

B. OPTIMAL CONTROL

Since the number of paths the robot could follow is infinite, some paths would be more efficient than others. In order to determine the most efficient path, we must first chose a cost function or a performance index. Following the development in Reference 5, pp. 180-183, for optimal control of a standard nonlinear system, we may obtain the necessary conditions for optimality.

We first express the differential equations of motion in the form,

$$\dot{x} = f(x, u, t) \tag{4}$$

The cost function can take the form

$$J = \Phi[x(t_f), t_f] + \int_{t_0}^{t_f} F[x(t), u(t), t]\, dt \tag{5}$$

where F could represent the pseudo-kinetic energy in the form u^2, with u as the velocity. The term Φ is a terminal cost vector and is a function of the states at the final time. This final time is not specified. Applying Lagrange multiplier vector, λ, we form the augmented functional.

$$J = \Phi + \int_{t_0}^{t_f} [F + \lambda^T (f - \dot{x})]\, dt \tag{6}$$

After defining the Hamiltonian as

$$H \equiv F + \lambda^T f \tag{7}$$

we can determine the necessary conditions for an optimal solution using Pontryagin's Minimization Principle: [Ref. 5:p. 183]

$$[H+\frac{\partial\Phi}{\partial t}]\,|_{t_f}\delta t_f=0 \tag{8}$$

$$[\lambda^T-\frac{\partial\Phi}{\partial x}]\,|_{t_f}\delta x(t_f)\,]=0 \tag{9}$$

$$\dot{\lambda}=-[\frac{\partial H}{\partial x}]^T \tag{10}$$

$$\dot{x}=[\frac{\partial H}{\partial\lambda}]^T\equiv f \tag{11}$$

$$\frac{\partial H}{\partial u}=0 \tag{12}$$

"The optimal control u(t) is determined at each instant to render the Hamiltonian a minimum over all admissible control functions." [Ref. 5:p. 183] Using the last condition, we can solve for the control input, u, in terms of the states, x, and what we will now refer to as costates, λ.

Let us now consider some simplifications to the above necessary conditions. If we fix the final time to achieve the desired condition, the first criterion is immediately satisfied as $\delta t_f = 0$. If we also describe the desired condition directly in terms of the states, x, and fix the value of the desired final states, then $\delta x(t_f) = 0$. Consequently, the second condition is met. In practical terms, this translates to going to a desired set of states in a fixed amount of time.

We now apply these simplifications to the differential expressions for the states and costates. Assuming our initial states are known, we have boundary conditions for the states at the initial and final time. However, we know neither the initial or final boundary conditions for the costates. And since the state and costate differential expressions are coupled, they must be solved simultaneously. As a result, the combined expressions give the form for a two point boundary value problem.

$$\begin{bmatrix} \dot{\vec{X}} \\ \dot{\vec{\lambda}} \end{bmatrix} = \begin{bmatrix} f(X,\lambda) \\ g(X,\lambda) \end{bmatrix} \qquad \begin{matrix} \vec{X}_{(t=t_0)} = \vec{X}_0 \\ \vec{X}_{(t=t_f)} = \vec{X}_f \end{matrix} \tag{13}$$

C. TWO POINT BOUNDARY VALUE PROBLEMS

In the case of linear differential equations, many analytic methods are available for solution of two point boundary value problems. However for nonlinear problems like the mobile robot, analytic methods for the solution to the two point boundary value problem do no exist. In some cases, non-linear problems can be solved analytically. Such problems are generally very simple and may only represent special cases of an overall problem. As we shall see later, the mobile robot problem does not lend itself readily to analytic methods.

In many cases, a non-linear two point boundary value problem can best be solved numerically. Unfortunately, numerical methods for non-linear two point boundary value problems are usually fairly complicated.

The general approach is to make an initial "guess" to the solution and adjust this trial solution to match the boundary conditions and differential equations. There are two distinct methods for solving such problems, shooting and relaxation [Ref. 6].

The first method, shooting, requires an initial guess of dependent variables based upon one boundary. Then using numerical methods common to initial value problems, we obtain a trial solution. This trial solution is compared against the second boundary. The error between the two is noted and the free parameters of the equation adjusted accordingly. This repeats until the error is sufficiently small. The advantages of this method are its simplicity and relative speed. For extremely non-linear systems, however, systematically improving the solution can prove difficult.

In the second method, relaxation, the differential equations are converted into difference expressions using Taylor series expansion. With an arbitrary initial trial solution, the variance of each point in the discretized mesh is calculated. The trial solution is then adjusted to improve agreement with the differential equations and the boundary conditions. This continues iteratively until the variance, or error, of the solution is sufficiently small. Relaxation methods are considered advantageous for problems with complicated boundary conditions, but smooth and non-oscillatory functions. Two disadvantages of this method are the large number of variables to be solved simultaneously and complexity of the expressions required in the algorithm. The number of differential equations, mesh size and coupling of adjacent points in the mesh determine the number of variables to solve. For example, in a system with 8 differential equations, on a mesh with 100 points, coupling two points, 1600 variables would result.

For the mobile robot problem, the kinematic equations involve trigonometric functions. As we shall see in Chapter III, the resulting state and costate differential equations are highly non-linear. In anticipation of the highly non-linear kinematic behavior of a mobile robot, the approach taken here is the relaxation method. We take advantage of published computer programs designed specifically for this method.

D. APPLICATION OF THE RELAXATION METHOD

As previously stated, the method starts with an initial guess trajectory for each of the differential equations and then adjusts these trial solutions to match both the governing equations and the boundary conditions. The method in which the computer program makes the corrections to the trajectories is a key to finding a proper solution. The source of the computer code and expression preparation process used here is Reference 6.

Given a set of N coupled first order differential equations, we first divide the independent variable domain into M discrete mesh points, t_k, $k = 1,2,..M$. For our problem, the initial state boundary values are located at t_1 and the final state boundary values at t_M. The costate boundary values are not fixed. The N differential equations then become finite difference equations to describe the internal meshpoints. We define the vector y_k as the entire set of dependent variables at point t_k. The exact form of the finite difference equation depends on the coupling desired. For our purposes, a backward difference technique is sufficient. This will couple each point on the mesh with the point preceding it.

By comparing the difference between adjacent solution values, $(y_k - y_{k-1})$, to the solution of the finite difference equations, we form an error equation. A solution exists where the error equations are zero and the boundary conditions are met. Considering any internal mesh point, k, this error expression takes the form

$$E_k = y_k - y_{k-1} - (t_k - t_{k-1})\, g_k(t_k, t_{k-1}, y_k, y_{k-1}) \tag{14}$$

Through Taylor series expansion of the error equation we determine the variance of the error with small changes in Δy_k. Since we are looking for the solution where the error is zero, for the internal mesh points, $k=2,3...M$, the form is

$$\sum_{n=1}^{N} S_{j,n}\Delta y_{n,k-1} + \sum_{n=N+1}^{2N} S_{j,n}\Delta y_{n-N,k} = -E_{j,k} \tag{15}$$

$$j=1,2,\ldots N$$

where

$$S_{j,n}=\frac{\partial E_{j,k}}{\partial y_{n,k-1}}, \quad S_{j,n+N}=\frac{\partial E_{j,k}}{\partial y_{n,k}}, \tag{16}$$

$$n=1,2,\ldots,N$$

At each internal point, k, $S_{j,n}$ forms a N X $2N$ matrix. The contents of this matrix are corrections to the solution variables between points k and $k-1$.

At the initial boundary, since E_1 depends only on y_1 the relation takes the form

$$\sum_{n=1}^{N} S_{j,n}\Delta y_{n,1}=-E_{j,1}, \tag{17}$$

$$j=n_2+1, n_2+2, \ldots, N$$

where

$$S_{j,n}=\frac{\partial E_{j,1}}{\partial y_{n,1}}, \tag{18}$$

$$n=1,2,\ldots,N$$

And similarly, at the final boundary, where E_M depends only on y_M, the form is

$$\sum_{n=1}^{N} S_{j,n}\Delta y_{n,M}=-E_{j,M+1}, \tag{19}$$

$$j=1,2,\ldots,n_2$$

where

$$S_{j,n}=\frac{\partial E_{j,M+1}}{\partial y_{n,M}}, \tag{20}$$

$$n=1,2,\ldots,N$$

The above equations can now be used to solve for corrections, Δy, to the trial solution vector, y. This process continues iteratively until the correction are sufficiently small. Of course, since the equations are coupled, they must be solved simultaneously.

If we combine the expressions for each internal point and boundary points in a global matrix, we see that matrix has a special "block diagonal" form (Fig. 2). This form allows a more economical matrix inversion process. The matrix inversion is accomplished through a form of Gaussian elimination which takes advantage of the special form. This process requires significantly less computational time or storage than inversion of the entire matrix. This is critical due to the size of the global matrix, (MN X MN).

Recall that our overall goal is to determine the optimal trajectory for a mobile robot traversing from one position and orientation to another. Application of optimal control theory results in a two point boundary value problem. Using the method described above, we can solve most problems of this form. However, this method does not guarantee a solution. Many factors will affect the program's ability to converge to a solution. Therefore before attempting the two wheeled mobile robot problem, a simpler related problem will be solved. This will serve to provide insight on use of the program and validate the process.

II. OPTIMAL CONTROL OF A ROLLING DISK

In Chapter I, we provided an outline for the optimal control problem of a dynamical system. In this chapter we apply Pontryagin's Minimization Principle [Ref. 5] and solve the associated two point boundary value problem for the simple example of a rolling disk. The differential equations of motion for the disk and robot systems are similar, and the nonholonomic constraint is exactly the same; no side slipping is allowed. The only difference between the rolling disk and the mobile robot model is the addition of a state variable: the angular orientation of the rolling disk about it's rotational axis, ϕ.

A. PROBLEM DESCRIPTION

Consider a vertical disk rolling on the horizontal, X-Y plane. (Fig. 3). Like the mobile robot, the orientation of this disk with respect to the plane will be described as an angular displacement, θ, from the X axis. The orientation of the disk face with respect to it's axis of rotation is described as an angular displacement, ϕ, from the normal vector to the X-Y plane. This gives us a total of 5 coordinates to describe the position and orientation of the disk.

The velocity of the disk, like the robot, can be described in terms of translation and rotation. The translational velocity again is constrained to the direction of orientation of the disk. However, the forward velocity of the disk is directly related to the angular velocity of ϕ and disk radius, R. If we consider the variation of θ and ϕ with time as control inputs, U_1 and U_2 respectively, the state-space form of the kinematic equations becomes

$$\begin{Bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \\ \dot{\phi} \end{Bmatrix} = \begin{bmatrix} 0 & R\cos\theta \\ 0 & R\sin\theta \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \end{Bmatrix} \tag{21}$$

or in a more condensed form as

$$\dot{\vec{X}} = [K]\ \vec{u} \tag{22}$$

B. OPTIMAL CONTROL

The objective for this problem is to roll the disk from and initial position and orientation to a desired final position and orientation in some optimal manner. Note that for our problem the time to accomplish this task is fixed. The choice of units for the X-Y parameters are arbitrary. The angular displacements are in non-dimensional radians. Time is considered on the unity scale with 0 at t_0 to 1 at t_f. The initial conditions at t_0 are defined as X_i, Y_i, θ_i, ϕ_i, and the final conditions at t_f as X_f, Y_f, θ_f, ϕ_f.

The development of the optimal control problem follows the method described in Chapter I. To determine an optimal path for the disk, we define the performance parameter as

$$J = \frac{1}{2}\int_{t_0}^{t_f} u^T u \, dt \tag{23}$$

Since the terminal costs are path independent, they are neglected here. Adding a Lagrange multiplier the cost functional becomes

$$J = \int_{t_0}^{t_f} \left(\frac{1}{2}u^T u + \lambda^T(Ku - \dot{x})\right) dt \tag{24}$$

By defining the Hamiltonian,

$$H = \frac{1}{2}(u^T u + \lambda^T K u) \tag{25}$$

the optimal control is obtained as:

$$u = -K^T\lambda \tag{26}$$

Substituting this expression into equation (25), the Hamiltonian becomes

$$H = -\frac{1}{2}(\lambda^T K K^T \lambda) \tag{27}$$

Using this new expression, the states can be expressed as

$$\dot{X} = -K K^T \lambda \tag{28}$$

or in expanded form,

$$\begin{Bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \\ \dot{\phi} \end{Bmatrix} = \begin{bmatrix} -\frac{R^2\lambda_1}{2}(1 + \cos 2\theta) - \frac{R^2\lambda_2}{2}\sin 2\theta - R\cos\theta\lambda_4 \\ \frac{-R^2\lambda_1}{2}\sin 2\theta - \frac{R^2\lambda_2}{2}(1 - \cos 2\theta) - R\sin\theta\lambda_4 \\ -\lambda_3 \\ -(R\cos\theta\lambda_1 + R\sin\theta\lambda_2 + \lambda_4) \end{bmatrix} \tag{29}$$

Similarly, the costates equations can be expressed as

$$\dot{\lambda} = -\frac{\partial}{\partial x}\left(-\frac{1}{2}\lambda^T K K^T \lambda\right) \tag{30}$$

Noting that the matrix K is only a function of state variable θ, the individual costate equations become

$$\begin{aligned} \dot{\lambda}_1 &= 0 \\ \dot{\lambda}_2 &= 0 \\ \dot{\lambda}_3 &= \left(\begin{aligned} &\frac{R^2}{2}[(\lambda_2^2 - \lambda_1^2)(\sin 2\theta + 2\lambda_1\lambda_2 + \cos 2\theta)] \\ &\qquad +R\lambda_4(\lambda_2\cos\theta - \lambda_1\sin\theta) \end{aligned} \right) \\ \dot{\lambda}_4 &= 0 \end{aligned} \tag{31}$$

Combining the states and costates into a single vector gives the structure for the two point boundary value problem.

$$\begin{Bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{\lambda}_1 \\ \dot{\lambda}_2 \\ \dot{\lambda}_3 \\ \dot{\lambda}_4 \end{Bmatrix} = \begin{bmatrix} \left[-\frac{R^2\lambda_1}{2}(1 + \cos 2\theta) - \frac{R^2\lambda_2}{2}\sin 2\theta - R\cos\theta\,\lambda_4\right] \\ \left[-\frac{R^2\lambda_1}{2}\sin 2\theta - \frac{R^2\lambda_2}{2}(1 - \cos 2\theta) - R\sin\theta\,\lambda_4\right] \\ \left[-\lambda_3\right] \\ \left[-(R\cos\theta\lambda_1 + R\sin\theta\lambda_2 + \lambda_4)\right] \\ 0 \\ 0 \\ \begin{bmatrix} \frac{R^2}{2}\left[(\lambda_2^2 - \lambda_1^2)\sin 2\theta + 2\lambda_1\lambda_2 + \cos 2\theta\right] + \\ R\lambda_4(\lambda_2\cos\theta - \lambda_1\sin\theta) \end{bmatrix} \\ 0 \end{bmatrix} \tag{32}$$

To the best of our knowledge, no analytical solution exists to this problem. A similar problem has been solved analytically by Cameron [Ref. 7]. However, his problem looks for the minimum time solution. By use of Pontryagin's Minimization Principle, equation (12), this implies use of the time derivative of the Hamiltonian. For the minimum time problem, it can be shown that the Hamiltonian is a constant. However in our problem, the final time is fixed and terminal cost, Φ, is zero. From equation (8), the Hamiltonian may therefore be any value over time. Therefore, Cameron's analytical method does not apply to our fixed time problem.

C. NUMERICAL SOLUTION BY THE RELAXATION METHOD

Given the N differential equations above, we apply the relaxation method described in Chapter I to develop expressions required by the relaxation method computer program. This essential entails finding the elements of the S matrix. For the interior meshpoints, a total of N X $2N$ such expressions must be developed. The two boundaries each require an additional N X N expressions. Since there are eight differential equations, we must develop a total of 144 $S_{j,n}$ expressions. Fortunately, many of the expressions for this particular problem will turn out to be zero.

Rather than repeating the development for all these expressions here, an example of developing expressions for an interior point is presented. Given a differential equation which describes the interior mesh points, the first step is to substitute Y_n for all dependent variables, where n is the equation number, such that

$$\begin{array}{c} X \rightarrow Y_1 \\ Y \rightarrow Y_2 \\ \theta \rightarrow Y_3 \\ \phi \rightarrow Y_4 \\ \lambda_1 \rightarrow Y_5 \\ \lambda_2 \rightarrow Y_6 \\ \lambda_3 \rightarrow Y_7 \\ \lambda_4 \rightarrow Y_8 \end{array} \tag{33}$$

We then apply the finite difference expression

$$Y_{n,k} = \frac{Y_{n,k} + Y_{n,k-1}}{2} \tag{34}$$

to each independent variable. Taking the first state equation as an example, the finite difference equation is

$$\dot{Y}_1 = \left\{ \begin{array}{l} \dfrac{-R^2}{2}\dfrac{(Y_{5,k}+Y_{5,k-1})}{2}\left(1+\cos 2\left(\dfrac{Y_{3,k}+Y_{3,k-1}}{2}\right)\right) \\ -\dfrac{R^2}{2}\dfrac{(Y_{6,k}+Y_{6,k-1})}{2}\left(\sin 2\left(\dfrac{Y_{3,k}+Y_{3,k-1}}{2}\right)\right) \\ -R\dfrac{(Y_{8,k}+Y_{8,k-1})}{2}\cos\left(\dfrac{Y_{3,k}+Y_{3,k-1}}{2}\right) \end{array} \right\} \tag{35}$$

Next, the finite difference equation is placed into the error expression.

$$E_{1,k} = (Y_{1,k}-Y_{1,k-1}) - h \left\{ \begin{array}{l} \dfrac{-R^2}{2}\dfrac{(Y_{5,k}+Y_{5,k-1})}{2}\left(1+\cos 2\left(\dfrac{Y_{3,k}+Y_{3,k-1}}{2}\right)\right) \\ -\dfrac{R^2}{2}\dfrac{(Y_{6,k}+Y_{6,k-1})}{2}\left(\sin 2\left(\dfrac{Y_{3,k}+Y_{3,k-1}}{2}\right)\right) \\ -R\dfrac{(Y_{8,k}+Y_{8,k-1})}{2}\cos\left(\dfrac{Y_{3,k}+Y_{3,k-1}}{2}\right) \end{array} \right\} \tag{36}$$

Where h is the grid spacing on the mesh. For our evenly spaced mesh,

$$h = \frac{1}{M-1} \tag{37}$$

As given by equation (16), the $S_{j,n}$ expressions are the partial derivatives of the error expressions with respect to each of the states and costates at meshpoint k and k-1.

Again, in the interest of brevity, only two $S_{j,n}$ expressions are presented here. Taking $S_{1,5}$ as the first example yields

$$S_{1,5} \equiv \frac{\partial E_{1,k}}{\partial Y_{5,k-1}} = \frac{hR^2}{4}(1+Cos(Y_{3,k}+Y_{3,k-1})) \tag{38}$$

Fortunately, due to the finite difference method chosen, these expressions tend to repeat. For example $S_{1,13}$ yields

$$S_{1,13} \equiv \frac{\partial E_{1,k}}{\partial Y_{5,k}} = \frac{hR^2}{4}(1+Cos(Y_{3,k}+Y_{3,k-1})) \tag{39}$$

the same as $S_{1,5}$. The development of the other 126 interior meshpoint expressions follow similarly, some simpler than others. The final result for all of these terms can be seen in the DIFEQ.FOR subroutine in Appendix A.

The expressions for the boundary expressions, though similar, fall under equations (18) and (20). The major difference for the boundary expressions is that they are not based on the differential equations. Since our boundary conditions are simply state values, the error expressions are at the initial and final time are of the form

$$\begin{aligned} E_{n,1} &= Y_{n,1} \\ E_{n,M+1} &= Y_{n,M} \end{aligned} \tag{40}$$

where n is the nth variable as given by equation (33). Thus for the initial and final boundary conditions respectively,

$$\left\{ \frac{\partial E_{j,1}}{\partial Y_{n,1}} \equiv S_{j+n_1,n+N} \right\} \atop \left\{ \frac{\partial E_{j,M+1}}{\partial Y_{n,M}} \equiv S_{j,n+N} \right\} = \left\{ \begin{matrix} 1, & for\ j=n \\ 0, & for\ j\neq n \end{matrix} \right\} \qquad \textbf{(41)}$$

$$(j=1,2,\ldots,N)$$
$$(n=1,2,\ldots,N)$$

where N is the total number equations and n_1 is the number of boundary conditions at the initial time. The shift in indices by N and n_1 is necessary to take advantage of the 'block diagonal' form of the overall matrix of S expressions. The result is the unity matrix for the initial and final S expressions. Note that for more complicated boundary conditions, such as a manifold of states or terminal costs, the relationships above are not valid.

Next, we must develop an initial guess for the values of the states and costates for all points on the mesh. For the states this guess can be somewhat intuitive. For example, we desire that the disk start at the X,Y position (0,0) and roll ten (10) times and make one (1) complete turn to return to the starting position. Therefore, the initial and final boundary conditions are

$$\begin{array}{ll} X_0 = 0 & X_f = 0 \\ Y_0 = 0 & Y_f = 0 \\ \theta_0 = 0 & \theta_f = 2\pi \\ \phi_0 = 0 & \phi_f = 20\pi \end{array} \qquad \textbf{(42)}$$

where the angular terms are expressed in radians. Intuitively, we would expect the most optimal path in the X-Y

plane to be a circle. If we initially assume that the angular terms vary at a constant rate, the initial guess trajectory for the state variables will appear as shown in Figure 4. Since we have no information on the costate behavior, we will assume the initial trajectory for each costate to be a constant value of zero.

After the required expressions and initial guess entry method is successfully compiled, the program is ready to run. A sampling of the results follow.

D. DISCUSSION OF RESULTS

For the case described above, the program converges in a few hundred iterations. From Figure 5, we see that the final state solution is in fact the same as the initial guess. The iterations were required to adjust the costate solutions to their proper trajectories. (Fig. 6) Since the state solution gives the expected circular path, the solution appears to be optimal.

For a more rigorous validation, we substitute the costate solutions

$$\begin{aligned} \lambda_1(t) &= 0 \\ \lambda_2(t) &= 0 \\ \lambda_3(t) &= -2\pi \\ \lambda_4(t) &= -20\pi \end{aligned} \qquad \textbf{(43)}$$

back into equation (32).

The derivative equations can then be expressed as

$$
\begin{aligned}
\dot{X} &= 20\pi\cos\theta \\
\dot{Y} &= 20\pi\sin\theta \\
\dot{\theta} &= 2\pi \\
\dot{\phi} &= 20\pi \\
\dot{\lambda}_1 &= 0 \\
\dot{\lambda}_2 &= 0 \\
\dot{\lambda}_3 &= 0 \\
\dot{\lambda}_4 &= 0
\end{aligned}
\tag{44}
$$

Note that the angular velocity terms are constant. This is consistent with the minimization of our cost function. And since this is a kinematics problem, the velocities may be non-zero at the initial and final time. Integrating the state terms yields

$$
\begin{aligned}
X(t) &= 10\sin\theta \\
Y(t) &= 10(1-\cos\theta) \\
\theta(t) &= 2\pi t \\
\phi(t) &= 20\pi t
\end{aligned}
\tag{45}
$$

which gives the equation for a circle in the X-Y plane.

If we make a slightly different initial guess for the states, such as an ellipse (Fig. 7) the final result is the same. If however, the initial guess is not sufficiently good, the program does not converge. While the initial guess for the states can usually be based on some intuitive reasoning, providing a sufficiently good estimate of the costate can prove difficult. For this problem, an initial guess of all zeros for costates works quite well. If, however, we chose trial values that are 10 units away from the proper solution, the program does not converge. Thus, while the costates may not be particularly important to the usable state space solution, they are necessary to solve the optimal control problem. Generally though, a poor estimation of the costate can be compensated for by a good state estimation.

Where the circular path presents a fairly simple solution, we now choose a more difficult task for our disk. This will demonstrate the usefulness of this method for problems where the optimal path is not obvious. For example, we desire that the disk make 10 rolls and 5.5 turns while moving on the X-Y axis form a point (10,10) to a point (-5,-2). As an initial guess, we shall use the state and costate solution to the circular problem above. The resulting path, obtained after several hundred iterations, appears in Figure 8. The state and costate trajectories appear in Figures 9 and 10, respectively. While these solutions appear optimal, they are not obvious at the outset of the problem.

The program for the disk problem has been tested extensively and, when provided a sufficiently good guess, found to give an apparently optimal solution for all cases except one. For the case of rolling the disk where the initial and final θ boundary conditions are the same and lie along the same line, the program does not converge. However, the program will converge if there is at least a very small difference between the initial and final angles. For the nearly straight line case, the smallest angular difference which results in a convergence is .00036 degrees. (Fig. 11). To achieve this it is necessary reduce the SLOWC program parameter to cause smaller adjustments to the trial solution. This indicates that for small difference in boundary θ values, the program is sensitive to small changes. If we exaggerate the distance the disk must roll between these two points, the reason for this behavior becomes evident. (Fig. 12) Here we specified that the disk roll 5 times. The initial θ value is 45 degrees. If we require that the disk make a 3.6×10^{-8} degree turn to the left (1×10^{-10} rotation) we obtain one optimal solution. However, if we require that the disk make a 3.6×10^{-8} degree turn to the right (-1×10^{-10} rotation) we obtain a much different solution. Hence, for very small changes in angle the solution varies widely. If we specify a zero rotation, there is no clear preference for the most optimal solution, and the program cannot converge. The same holds true as we approach a perfectly straight line path. We specify the initial and final position and the initial and final ϕ values, which theoretically are the same. However numerically, there is a small difference. This difference is sufficient to induce the problems above and prevent convergence. Fortunately, an analytic solution to the exact straight line problem is easily obtained.

The computer program used includes several control parameters which assist in finding a solution. While running various simulations, the following trends were noted:

- Slowing down the convergence by decreasing SLOWC can help find a solution when the maximum error fluctuates near some minimum value. However, doing so does not guarantee a solution.

- The SCALV values should represent the absolute magnitude of a typical solution value. Where this value is not known, use a small SCALV to start.

- The trend of the maximum error with iterations should be used as a guide as to whether the program will converge to a solution. However, the trend neither guarantees nor excludes convergence.

As demonstrated, the rolling disk problem requires a path which is continuous and smooth. And since we specify the distance the disk must roll and number of turns the disk must make, the solution is only optimal for those specifications. In the more general mobile robot problem, we look for the most optimal path which need only meet the initial and final boundary conditions.

III. OPTIMAL CONTROL OF A MOBILE ROBOT

In Chapter II we developed a two point boundary value problem by applying Pontryagin's Minimization Principle to the equations of motion for a rolling disk. We then solved the resulting two point boundary value problem by a numerical relaxation technique. Having demonstrated that the process above provides an optimal solution for a simple nonholonomic system, we return to the more difficult mobile robot problem. Our goal is to move the robot from a initial position and orientation to a desired one within a fixed amount of time, in an optimal manner, and using feedback control. Before developing our solution, however, we first look at some conventional theory regarding mobile robot control. This is necessary to motivate the approach used to solve our problem.

A. LITERATURE SURVEY

Extensive research into non-linear control design of two wheeled mobile robots exists. For the problems of path following and tracking, relatively classical non-linear control techniques have been applied successfully. [Refs. 2,3] However, the problem of stabilization about a point is more difficult. Brockett's Theorem [Ref. 8] shows that smooth non-time varying control laws cannot be developed for such problems.

This is the case for all driftless, nonholonomic systems of the form

$$\dot{\vec{X}} = [K]\ \vec{u} \tag{46}$$

Using classical Lyapunov analysis, Reference 1 presents a general method for finding time varying control laws for driftless systems. In Reference 6 and 7, the authors develop smooth, time varying and piecewise continuous control laws. While these controls employ closed loop feedback, none considers the optimality of the solution. In this research, we apply optimal control theory to the mobile robot problem.

B. STATE AND COSTATE EQUATIONS FOR THE MOBILE ROBOT

In our approach to the mobile robot control problem we first move the robot onto the line described by the final position and orientation of the robot. (Fig. 13) The robot may then roll directly to the final desired position. The point at which the robot will intersect the line and the manner in which the robot will approach the line is not specified. The goal of the optimal control problem is to minimize the distance between the desired position of the robot and the point of intersection of the robot with the line, in a way that utilizes the minimum amount of energy.

1. Basic Kinematic Relationships

Returning to the coordinate and velocity descriptions of Figure 1, we begin with the kinematic equations,

$$\begin{aligned} \dot{X} &= U_1 \cos\theta \\ \dot{Y} &= U_1 \sin\theta \\ \dot{\theta} &= U_2 \end{aligned} \tag{47}$$

From the desired final conditions of X_d, Y_d, and θ_d, we redefine our states in terms of the difference between the final condition and the current coordinate value such that

$$\begin{aligned} \Delta X &= X_d - X \\ \Delta Y &= Y_d - Y \\ \Delta\theta &= \theta_d - \theta \end{aligned} \tag{48}$$

As our approach suggests, we require that the difference between the robot angle and desired angle be minimized or,

$$\Delta\theta = 0 \tag{49}$$

We also require that the perpendicular distance between the robot and the line be minimized. This distance can be defined in trigonometric terms as

$$p \equiv (\Delta Y \cos\theta_d - \Delta X \sin\theta_d) \tag{50}$$

In order to converge p and $\Delta\theta$ to zero asymptotically, we find that the second input should be a function of the first input. Our analysis is based on the application of Lyapunov's Stability Theorem.

2. Application of Lyapunov's Theorem

Lyapunov's Theorem of asymptotic stability provides that the equilibrium of zero for a system,

$$\dot{\vec{x}} = f(\vec{x}, \vec{u}) \tag{51}$$

is asymptotically stable if there exists a positive definite function such that the first derivative of that function is non-increasing. [Ref. 9] In our case we define a Lyapunov function as

$$V = \tfrac{1}{2}(p^2 + \Delta\theta^2) \tag{52}$$

The first derivative of this function is,

$$\begin{aligned} \dot{V} &= -p(\cos\theta_d \sin\theta - \sin\theta_d \cos\theta)\, U_1 - \Delta\theta\, U_2 \\ &= pU_1 \sin(\Delta\theta) - \Delta\theta\, U_2 \\ &= -\Delta\theta\left(U_2 - pU_1 \frac{\sin(\Delta\theta)}{\Delta\theta}\right) \\ &= -\Delta\theta\,(U_2 - pU_1 f(\Delta\theta)) \end{aligned} \tag{53}$$

where

$$f(\Delta\theta) \triangleq \frac{\sin(\Delta\theta)}{\Delta\theta} \tag{54}$$

If we choose

$$U_2 = pU_1\, f(\Delta\theta) + \alpha\,\Delta\theta \tag{55}$$

We may express equation (53) as

$$\dot{V} = -\alpha\,\Delta\theta^2 \tag{56}$$

which is negative semidefinite.

This equation satisfies Lyapunov's Theorem for all α greater than zero, provided that $\Delta\theta$ is not equal to zero. In the event $\Delta\theta$ is equal to zero, the derivative of the Lyapunov function becomes negative semidefinite and the asymptotic stability can be guaranteed by applying the theorem by Mukherjee and Chen [Ref. 10].

The choice of the second control, U_2, given by equation (55), in terms of the first control, U_1, and state feedback leaves us with the task for the design of one input for the system, namely U_1. U_1 will be designed using optimal control methods. The gain, α, affects the rotational motion, and from Lyapunov's Theorem, α must be greater than zero at all times. Various schemes have been tested to determine the best use of this parameter in an optimal solution for U_1.

3. Variations of the Robot Problem

Since the only requirement of U_2 is that equation (53) be negative definite, there are infinite variations of this function which we could employ. In the sections below, we produce five possible variations and discuss the application of optimal control to each of them.

a. Robot 1, Virtual Robot Problem

In this approach, in addition to the original robot, we define a virtual robot which may travel only on the line of the desired angle. (Fig. 14) This approach is somewhat similar to the bi-directional approach. [Ref. 11] The virtual robot may roll forward or backward, but not turn.

Our goal is to have the two robots meet at some unspecified point on the line. This allows a smooth trajectory for each robot. Furthermore, this positions and orients the real robot in line with the desired final location, requiring only a trivial solution to complete. Fortunately, the impact of this addition to the kinematic equations is minimal. Defining the position of the virtual robot as X_d and Y_d, which are now variables, the difference between the two positions is

$$\begin{aligned} \Delta X &= (X_d - X) \\ \Delta Y &= (Y_d - Y) \end{aligned} \tag{57}$$

The difference between the orientation of the two robots is

$$\Delta\theta = (\theta_d - \theta) \tag{58}$$

where θ_d is the constant desired angle of orientation. The differential equations of motion now take the form,

$$\begin{Bmatrix} (\dot{\Delta X}) \\ (\dot{\Delta Y}) \\ (\dot{\Delta\theta}) \end{Bmatrix} = \begin{bmatrix} \cos\theta_d U_d - \cos(\theta_d - \Delta\theta)\, U_1 \\ \sin\theta_d U_d - \sin(\theta_d - \Delta\theta)\, U_1 \\ -\alpha\,\Delta\theta - pU_1 f(\Delta\theta) \end{bmatrix} \tag{59}$$

Where U_d is the forward/backward velocity of the virtual robot. Applying the same optimal control theory as before, we define our cost function as

$$J = \frac{C}{2}(\Delta X^2 + \Delta Y^2 + \Delta\theta^2)_{t_f} + \int_{t_0}^{t_f} \frac{1}{2}(U_1^2 + U_d^2) \tag{60}$$

The terminal cost gives a penalty for not going to the desired final condition, ΔX, ΔY, and $\Delta\theta$. C is the weighting parameter for this cost. This cost is necessary as the values of the final states tend to float and hence we cannot assume as before that

$$\delta\vec{X}(t_f)=0 \tag{61}$$

From equation (9) we see that to satisfy the necessary conditions for optimal control, we need

$$\lambda^T=\frac{\partial\phi}{\partial x}, \quad at\ t=t_f \tag{62}$$

where

$$\phi = \frac{1}{2}(\Delta X^2 + \Delta Y^2 + \Delta\theta^2) \tag{63}$$

This implies that the constraints at the final times are

$$\begin{aligned}(\lambda_1)_{t_f} &= C(\Delta X)_{t_f}\\ (\lambda_2)_{t_f} &= C(\Delta Y)_{t_f}\\ (\lambda_3)_{t_f} &= C(\Delta\theta)_{t_f}\end{aligned} \tag{64}$$

As we shall see later, this is important in minimizing the error at the final time. From the definition of the Hamiltonian,

$$H = L + \lambda^T f \tag{65}$$

where

$$L=\frac{1}{2}(U_1^2 + U_d^2) \tag{66}$$

and f is the right hand side of equation (59). We apply equation (12) for both U_1 and U_d. From this we can show that the optimal control inputs are,

$$U_1=\lambda_1\cos(\theta_d - \Delta\theta) + \lambda_2\sin(\theta_d - \Delta\theta) + \lambda_3 pf(\Delta\theta) \tag{67}$$

and

$$U_d = -\lambda_1\cos\theta_d - \lambda_2\sin\theta_d \tag{68}$$

Applying equations (10) and (11) we can develop a full set of equations for our two point boundary value problem.

$$\begin{Bmatrix} (\dot{\Delta X}) \\ (\dot{\Delta Y}) \\ (\dot{\Delta\theta}) \\ \dot{\lambda}_1 \\ \dot{\lambda}_2 \\ \dot{\lambda}_3 \end{Bmatrix} = \begin{bmatrix} \cos\theta_d U_d - \cos(\theta_d - \Delta\theta)\, U_1 \\ \sin\theta_d U_d - \sin(\theta_d - \Delta\theta)\, U_1 \\ -\alpha\Delta\theta - pU_1 f(\Delta\theta) \\ -\lambda_3 U_1 \sin\theta_d f(\Delta\theta) \\ \lambda_3 U_1 \cos\theta_d f(\Delta\theta) \\ \begin{bmatrix} \lambda_1 U_1 \sin(\theta_d - \Delta\theta) \\ - \lambda_2 U_1 \cos(\theta_d - \Delta\theta) \\ + \lambda_3(\alpha + pU_1 f'(\Delta\theta)) \end{bmatrix} \end{bmatrix} \tag{69}$$

where U_1, U_d and p, are as described above. The function $f(\Delta\theta)$ requires special handling due to the $\Delta\theta$ term in the denominator. By L'Hopital's rule we know

$$\lim_{\Delta\theta\to 0} f(\Delta\theta) = \lim_{\Delta\theta\to 0} \frac{\sin(\Delta\theta)}{\Delta\theta} = 1 \tag{70}$$

and

$$\lim_{\Delta\theta\to 0} f'(\Delta\theta) = \lim_{\Delta\theta\to 0} \frac{\cos(\Delta\theta)}{\Delta\theta} - \frac{\sin(\Delta\theta)}{\Delta\theta^2} = 0 \tag{71}$$

Therefore, to maintain continuity during numeric processing we define

$$f(\Delta\theta) = \begin{Bmatrix} \dfrac{\sin(\Delta\theta)}{\Delta\theta}, & for\ \Delta\theta\neq 0 \\ 1, & for\ \Delta\theta=0 \end{Bmatrix} \tag{72}$$

and

$$f'(\Delta\theta) = \begin{Bmatrix} \dfrac{\cos(\Delta\theta)}{\Delta\theta} - \dfrac{\sin(\Delta\theta)}{\Delta\theta^2}, & for\ \Delta\theta\neq 0 \\ 0, & for\ \Delta\theta=0 \end{Bmatrix} \tag{73}$$

From equation (69) the utility of a numeric solution to the two point boundary value problem becomes clear.

The process for setting up the $S_{j,n}$ expressions for the computer program is similar to the rolling disk problem, although more lengthy and involved. The final expressions can be found in the DIFEQ.FOR subroutine of Appendix B.

Upon testing the virtual robot problem, it became obvious that the discontinuous path was not a problem for the program. (Fig 15) A look at the velocity components explains why. (Fig 16) When broken into components the velocities are smooth. We also note that the non-zero velocities at the initial and final time are not a source of concern, as this is a kinematics problem. Furthermore, since this is a motion planning problem and not feedback control, a two robot model has no practical disadvantages. However, the next robot models we consider are based upon a single robot.

b. Robot 2, Single Mobile Robot

The kinematic equations of motion for this problem are similar to the equations for the two mobile robot except that X_d and Y_d are fixed. As a result,

$$\dot{U}_d=0 \tag{74}$$

and the kinematic equations appear as

$$\begin{Bmatrix} (\dot{\Delta X}) \\ (\dot{\Delta Y}) \\ (\dot{\Delta \theta}) \end{Bmatrix} = \begin{bmatrix} -\cos(\theta_d - \Delta\theta)\, U_1 \\ -\sin(\theta_d - \Delta\theta)\, U_1 \\ -\alpha\,\Delta\theta - pU_1 f(\Delta\theta) \end{bmatrix} \tag{75}$$

We define the cost function for this problem as

$$J = \frac{c}{2}(\Delta X^2 + \Delta Y^2 + \Delta\theta^2)_{t_f} + \int_{t_0}^{t_f} \frac{1}{2}(U_1^2) \tag{76}$$

which has the same terminal costs as the two robot problem. We again use the definition of the Hamiltonian

$$H = L + \lambda^T f \tag{77}$$

where

$$L = \frac{1}{2}(U_1^2) \tag{78}$$

and f is the right hand side of equation (75). Applying equation (12) for U_1 we can show that for optimal control,

$$U_1 = \lambda_1 \cos(\theta_d - \Delta\theta) + \lambda_2 \sin(\theta_d - \Delta\theta) + \lambda_3 p f(\Delta\theta) \tag{79}$$

Applying equations (10) and (11) we again develop the equations for our two point boundary value problem.

$$\begin{Bmatrix} (\dot{\Delta X}) \\ (\dot{\Delta Y}) \\ (\dot{\Delta\theta}) \\ \dot{\lambda}_1 \\ \dot{\lambda}_2 \\ \dot{\lambda}_3 \end{Bmatrix} = \begin{bmatrix} -\cos(\theta_d - \Delta\theta)\, U_1 \\ -\sin(\theta_d - \Delta\theta)\, U_1 \\ -\alpha\Delta\theta - pU_1 f(\Delta\theta) \\ -\lambda_3 U_1 \sin\theta_d f(\Delta\theta) \\ \lambda_3 U_1 \cos\theta_d f(\Delta\theta) \\ \begin{bmatrix} \lambda_1 U_1 \sin(\theta_d - \Delta\theta) \\ - \lambda_2 U_1 \cos(\theta_d - \Delta\theta) \\ + \lambda_3(\alpha + pU_1 f'(\Delta\theta)) \end{bmatrix} \end{bmatrix} \tag{80}$$

which is the same as equation (69) except for the first two differential equations. The resulting expressions can be seen in the DIFEQ.FOR subroutine of Appendix C:

As expected, the problem works adequately for apparently non-smooth paths. (Fig. 17, 18) However, for the case where we ask the robot to change only its angle of orientation, the solution given indicates that the robot only spins without moving forward. While this solution is indeed optimal, it is evident from the definition of angular velocity in equation (80) that the robot turns without moving.

c. Robot 3, Contrained Robot Model

To contrain the angular rotation to prevent rotation when the robot is stopped, we must ensure that U_2 is entirely a function of U_1. In order to meet Lyapunov's Theorem we must ensure that equation (53) is negative at all times. In order to meet both requirements, we chose U_2 of the form

$$U_2 = pU_1 f(\Delta\theta) + \alpha g(U_1)\Delta\theta \tag{81}$$

where

$$g(U_1) = \begin{Bmatrix} 1, & for U_1 \neq 0 \\ 0, & for U_2 \neq 0 \end{Bmatrix} \tag{82}$$

This expression guarantees that Lyapunov's Theorem is satisfied. Application of equation (12) results in the control,

$$U_1 = \lambda_1 \cos(\theta_d - \Delta\theta) + \lambda_2 \sin(\theta_d - \Delta\theta) + \lambda_3 p f(\Delta\theta) \tag{83}$$

which is the same control from Robot 2.

Applying the other necessary conditions for optimal control gives the equations for the two point boundary value problem.

$$\begin{Bmatrix} (\dot{\Delta X}) \\ (\dot{\Delta Y}) \\ (\dot{\Delta\theta}) \\ \dot{\lambda}_1 \\ \dot{\lambda}_2 \\ \dot{\lambda}_3 \end{Bmatrix} = \begin{bmatrix} -\cos(\theta_d - \Delta\theta)\, U_1 \\ -\sin(\theta_d - \Delta\theta)\, U_1 \\ -\alpha\,\Delta\theta\, g(U_1) - pU_1 f(\Delta\theta) \\ -\lambda_3 U_1 \sin\theta_d f(\Delta\theta) \\ \lambda_3 U_1 \cos\theta_d f(\Delta\theta) \\ \begin{bmatrix} \lambda_1 U_1 \sin(\theta_d - \Delta\theta) \\ -\lambda_2 U_1 \cos(\theta_d - \Delta\theta) \\ +\lambda_3(\alpha\, g(U_1) + pU_1 f'(\Delta\theta)) \end{bmatrix} \end{bmatrix} \tag{84}$$

The resulting expressions, obtained as before, can be seen in the DIFEQ.FOR subroutine in Appendix D.

The solutions for this new variation are, in most cases, the same as those from Robot 2. (Fig. 19) The most significant difference is for the case where we ask the robot to change only its angle of orientation. The solution is now a trivial one; the robot does not move. (Fig. 20) When ΔX and ΔY are zero, the quantity p is zero and thus,

$$U_1 = \lambda_1 Cos(\theta_d - \Delta\theta) + \lambda_2 Sin(\theta_d - \Delta\theta) \tag{85}$$

And if λ_1 and λ_2 are zero for all time, then

$$\begin{gathered} U_1 = 0 \\ g(U_1) = 0 \end{gathered} \tag{86}$$

and therefore,

$$
\begin{aligned}
U_1 &= 0 \\
g(U_1) &= 0 \\
\Delta\dot{X} &= 0 \\
\Delta\dot{Y} &= 0 \\
\Delta\dot{\theta} &= 0 \\
\dot{\lambda}_1 &= 0 \\
\dot{\lambda}_2 &= 0 \\
\dot{\lambda}_3 &= 0
\end{aligned}
\tag{87}
$$

The terminal costs are met since

$$\lambda_{3_{t_f}} = C\Delta\theta_{t_f} \tag{88}$$

where $\Delta\theta$ is fixed and the λ_3 term simply becomes a large enough constant to meet this constraint (Fig. 21). Heuristically, this says that the most efficient manner to achieve the desired final condition is not to go. Such results occur any time two of the three states, ΔX, ΔY or $\Delta\theta$, are equal to zero. In such cases where p or $\Delta\theta$ is zero at all times and U_1 becomes zero, the state equations of motions from equation (84) are all equal zero and give a trivial solution.

So far, we have chosen the value of α at the outset of the program. However, as we shall show later, α has a direct impact on the final solution.

d. *Robot 4, Robot 3 with High/Low α Control*

Defining α as a control is complicated by the fact that α must be positive to satisfy Lyapunov's Theorem. We therefore define the cost function

$$J = \int_{t_0}^{t_f} \left(\frac{1}{2} U_1^2 + \alpha\right) \tag{89}$$

where α is greater than zero for all t. The resulting Hamiltonian is

$$\begin{aligned} H = {} & \tfrac{1}{2} U_1^2 + \alpha \\ & -\lambda_1 \cos(\theta_d - \Delta\theta)\, U_1 \\ & -\lambda_2 \sin(\theta_d - \Delta\theta)\, U_1 \\ & -\lambda_3 (p U_1 f(\Delta\theta) + \alpha g(U_1)\Delta\theta) \end{aligned} \tag{90}$$

Applying equation (12) to U_1 we find the same result as before,

$$U_1 = \lambda_1 \cos(\theta_d - \Delta\theta) + \lambda_2 \sin(\theta_d - \Delta\theta) + \lambda_3 p f(\Delta\theta) \tag{91}$$

For the second control we consider only those terms in the Hamiltonian associated with α.

$$H_\alpha = \alpha(1 - \lambda_3 g(U_1)\Delta\theta) \tag{92}$$

In order to minimize the Hamiltonian and maintain a positive α we define

$$\alpha = \begin{Bmatrix} \alpha_{\min}, & for\ \beta > 0 \\ \alpha_{\max}, & for\ \beta \le 0 \end{Bmatrix} \tag{93}$$

where

$$\beta = 1 - \lambda_3 g(U_1)\Delta\theta \tag{94}$$

The resulting equations for the two point boundary value problem are the same as equation (84), except that the value of α depends on β. The resulting S expressions are listed in the DIFEQ.FOR program in Appendix E.

Using this variation of U_2, the program has difficulty converging in many cases. The non-linear nature of α is the source of this difficulty. (Fig. 22) In many cases, the converged solution is the same as Robot 3. Since there appears to be little advantage to this variation, we seek a more proportional α control.

e. *Robot 5, Robot 3 with Proportional α Control*

To develop a proportional α control, we start with a new cost function,

$$J = \int_{t_0}^{t_f} \frac{1}{2}(U_1^2 + \alpha^2) \tag{95}$$

We find that U_1 is the same as before. If we only consider the α terms then,

$$2H_\alpha = \tilde{H} = \alpha^2 - 2\lambda_3 \alpha\, g(U_1)\, \Delta\theta \tag{96}$$

However, since $g(U_1)$ equals zero for U_1 equal to zero, H_α is already a minimum when U_1 equals zero and α approaches the positive side of zero. Thus we only need consider the case where U_1 is not zero. In this case $g(U_1)$ is one and will be dropped in the remaining expressions. Factoring H_α, we find,

$$\tilde{H}_\alpha = (\alpha - \lambda_3 \Delta\theta)^2 - (\lambda_3 \Delta\theta)^2 \tag{97}$$

If we neglect the second part of this expression as it is not a function of α, then to minimize H_α,

$$\alpha = \begin{Bmatrix} \lambda_3 \Delta\theta, & for \quad \lambda_3 \Delta\theta > 0 \\ \alpha_{\min}, & for \quad \lambda_3 \Delta\theta \le 0 \end{Bmatrix} \tag{98}$$

where $\alpha_{\min}$ is some value greater than zero.

The resulting differential equations differ from equation (84) only in that alpha is now a function of λ_3 and $\Delta\theta$. This must be taken into account when developing the S expressions. The changes to the resulting S expressions can be seen in the DIFEQ.FOR subroutine in Appendix F.

In general, this variation gives better solutions than all other variations discussed. The proportional alpha control is more likely to converge and gives an apparently more optimal solution. It's tendency to converge is more well behaved than other variations. However, it still requires a certain amount of user interaction to set the value of α_{min} and other program parameters to a value which will achieve convergence. Trends and comparisons of this and other variations of the mobile robot problem is the subject of the following section.

C. DISCUSSION OF RESULTS

There are many factors which affect convergence, optimality and error of the final solution. Since each variation was designed with a slightly different intent, comparison is difficult. This section discusses general trends noted during extensive testing of the programs.

The state variable solutions to the two point boundary value problems are in terms of the difference between the current and desired value. For presentation, we convert these values into X, Y, and θ coordinates.

In the case of Robot 1 desired coordinates move. Therefore, we substitute our solution back into the differential equations to obtain velocity profiles. For Robot 1 only, we then use a crude trapezoidal integration to determine the values of X, Y, θ, X_d and Y_d at each time. There is a small error imposed by this integration which may appear in the path plots for Robot 1. The true final error all cases is taken directly from the solution values of ΔX, ΔY, and $\Delta\theta$ and is not affected by this integration. By true final error we refer to the difference between the final robot coordinates and the desired robot coordinates. For most cases this true final error can be made negligibly small by adjusting α and C.

For the other Robot variations the velocity profiles are obtained in the same way. However, since X_d and Y_d are fixed for these problems, determining the X, Y, and θ coordinates is simply a matter of subtraction. The final error shown in these problems is more representative of the true final error.

For the sake of comparison, the results of each variation was tested against a single pseudo-energy cost function.

$$J = \int_{t_0}^{t_f} \frac{1}{2}\left(U_1^2+\alpha^2\right) dt \tag{99}$$

This cost value has mixed units. For simplicty we consider the costs shown in the figures below in nondimensional units,

$$\frac{Length^2}{Time} \tag{100}$$

While Robot 5 was the only variation developed from this performance parameter, this is the most encompassing cost description. The terminal costs were not considered for this part as these are compared in the form of final error.

1. Effect of Varying α and C Parameters

For each variation of the Robot program, the effect of varying the rotational gain, α, and terminal cost weighting, C, is different. Rather than present all the possible variations here, some of the more significant trends are sampled.

The variation of α, strongly affects the ability of the program to converge as well as the optimality and error in the final solution. There is a range of α for which each program will converge for a given set of boundary conditions. A typical example of the effect of varying α can be seen in Figure 23. These are the paths given by Robot 1 for boundary conditions of

$$\begin{array}{ll} X_0 = 0 & X_f = 10 \\ Y_0 = 0 & Y_f = 10 \\ \theta_0 = 0° & \theta_f = 90° \end{array} \tag{101}$$

We must also look at the angular trajectory for these solutions. (Fig. 24) Note that for the extreme values of α, there is a larger error in the final solution. Also note that the paths are of different lengths, indicating that some α's give more optimal solutions than others. The energy cost plot (Fig. 25) shows the effect of α on this cost.

From Figures 23 and 25, an α value of 25 appears to give both a minimum final error and cost. However, the results for this program and set of boundary conditions cannot be used as a guideline for all programs or cases.

In the particular case of Robot 4, the use of α_{min} and α_{max} must be handled carefully. If the two values are greatly different, the program will have difficulty converging. If the two values are too close, there is no advantage to using this program.

Robot 5 tends to converge for a larger range of α values. In general, any α for which the other programs converge will usually work for Robot 5. However in many cases, Robot 5 allows a lower α, and a lower final energy cost. As a rule of thumb, the lowest α_{min} for which the program converges gives the most optimal solution.

The variation of C is more straightforward; the higher the value of C, the smaller the final error. (Figs. 26, 27) However, certain limits do apply to this guideline. If C is too high for a given set of boundary conditions, the program tends not to converge. There is also a price for this accuracy. (Fig. 28) In general a more accurate final solutions show a higher final cost.

2. Sample of Test Cases

The Robot programs have been tested for many different boundary conditions and, subject to user supplied parameters, give optimal solutions. The following six cases are representative of the results. For each program variation we provide the best known control parameters for that program and set of boundary conditions. In this way we compare the best result for each.

a. 90 Degree Turn Problem

For this case the boundary conditions are:

$$\begin{array}{ll} X_0 = 0 & X_f = 10 \\ Y_0 = 0 & Y_f = 10 \\ \theta_0 = 0^\circ & \theta_f = 90^\circ \end{array} \tag{102}$$

Most of the programs give a similar result for this problem, except Robot 1. (Fig. 29, 30) This problem requires a relatively high α to achieve convergence. As a result, the programs which use α as a control, Robot 4 & 5, show no advantage. (Fig. 31, 32) Although the Robot 1 solution takes a longer path, by our definition of cost, its solution is more optimal.

b. 30 Degree Angle Parking Problem

For this case the boundary conditions are:

$$\begin{array}{ll} X_0 = 0 & X_f = 0 \\ Y_0 = 0 & Y_f = 2 \\ \theta_0 = 0^\circ & \theta_f = 30^\circ \end{array} \tag{103}$$

Here, the effect of α control is more evident. (Fig. 33, 34) In the case of Robot 4, the program has a more difficult time converging because of the non-linear α. As a result, its solution is the least optimal. (Fig. 35) Robot 5, however, gives the best optimal solution. While Robot 5's final error is higher than some others, considering the order of magnitude of the final error, the difference is negligible.

c. *270 Degree Turn Problem*

This case is inherently not optimal. The angular displacement required could be achieved more easily by going to -90° instead. However, these boundary conditions provide a more demanding test.

$$\begin{array}{ll} X_0 = 0 & X_f = 0 \\ Y_0 = 0 & Y_f = 2 \\ \theta_0 = 0^\circ & \theta_f = 270^\circ \end{array} \qquad \textbf{(104)}$$

The programs overcome the angular displacement problem by stopping and backing part way though the maneuver. (Fig. 36, 37, 38) Based on the final error and energy cost, no program has any distinct advantage over the others for this maneuver. (Fig. 39)

d. *180 Degree Turn On A Point*

For all Robot programs using partial feedback, if two of the three state variables, ΔX, ΔY or $\Delta\theta$, have zero difference between their initial and final boundary conditions, the result will be the trivial, "don't go" solution. If, however, there is at least small difference between the initial and final boundary conditions for two of the three states, a non-trivial solution can be obtained. For this reason, we use a small difference between the initial and final X boundary conditions for this problem.

$$
\begin{array}{ll}
X_0 = 0 & X_f = -0.01 \\
Y_0 = 0 & Y_f = 0 \\
\theta_0 = 0^\circ & \theta_f = 180^\circ
\end{array}
\tag{105}
$$

Robot 1 and 2, which do not include partial feedback, simply turn and go to the desired position. (Fig. 40) The remaining solutions are all similar. For all cases the energy cost is the same, with similar final error for the feedback problems. (Fig. 41)

e. Parallel Parking Problem

This case proved the most difficult of any for the program to solve. Again, we must avoid the boundary conditions where two of the three boundary conditions have zero difference. Only Robot 5 was able to produce a solution with reasonable minimum error. (Fig. 42, 43, 44) This is because only Robot 5 supports proportional α control. The choice of α is critical to the resulting solution. Since α is a gain which affects the angular velocity, too high an α_{min} results in highly non-linear solutions. This is true even in the case of Robot 5. (Fig 45, 46, 47) Nevertheless, a judicious choice of α_{min} results in an optimal and low final error solution.

f. Trivial Straight Line Case

For the trivial case where we ask the robot to move along a straight line from one position to another, the solution converges very quickly to the obvious solution. A few iterations are necessary to bring the costate variables to their proper values. These programs have no problem converging, unlike the disk problem, because there are no competing boundary conditions.

3. Other Trends Noted

The effect on of the initial guess cannot be overstated. For each case, the initial guess was based on the straight line path between the initial and final points. (Fig. 48) The costates were assumed to be some small, non-zero value. The Robot programs show strong tendency towards convergence even when given such a crude guess.

In some cases, particularly highly oscillatory solutions about small values, the program will tend toward convergence but then hover at some error value, or oscillate between two small error values. In these cases the best response is to adjust the SLOWC parameter to slow the program convergence. This causes the program to make smaller corrections where it might be jumping, back and forth, over a solution. Doing this does not guarantee a solution, but it is helpful in some cases.

The Program uses an EPSILON variable to determine the value of $f(\Delta\theta)$ when $\Delta\theta$ is nearly equal to zero. If $\Delta\theta$ is less than EPSILON, we consider it sufficiently close to zero to use the definition of $f(\Delta\theta)$ equal to one and $f'(\Delta\theta)$ equal to zero. From experimenting with the programs, any reasonably small value for this variable will give the same solution. This is true as $\Delta\theta$ rarely approaches zero within a solution, except at the final time. At the final time, the computer determines the values of the S_{ij} expressions based on the final boundary condition expressions. Since the function $f(\Delta\theta)$ does not appear in these expressions, the value of EPSILON has no effect. In the case where $\Delta\theta$ is less than EPSILON at some other time, the impact appears negligible for all reasonable values of EPSILON.

The contrained robot problems use the control of equation (82). The program uses the variable EPSILON2 to determine if the value of U_1 is sufficiently close to zero for $g(U_1)$ to be equal to zero. The value used for this EPSILON2 has been found to make little difference in the solution as U_1 is rarely zero for any length of time. (Fig. 49) Where the value of U_1 is less than EPSILON2 at some time, the solution profile tends to be non-smooth, making it difficult (though not impossible) for the program to converge. On the next iteration, the time location of the zero U_1 may be moved. As a result, the program solution tends to place the exactly zero U_1 velocities between the discrete time points.

Hence, for reasonably sized values of EPSILON2, the solution is not affected significantly. However, for convergence sake, the value of EPSILON2 should be sufficiently small, 1×10^{-4} or less.

IV. SUMMARY AND RECOMMENDATIONS

In this thesis, we have demonstrated a method for finding an optimal, open loop, time varying control for a nonholonomic system. In general this method employs Pontryagin's Minimization Principle to find the state and costate equations for an optimal control. Then a numerical relaxation technique is applied to the resulting two point boundary value problem. For the specific problem of a two wheeled mobile robot, we first develop a partial feedback law using Lyapunov's Theorem. In doing so, we create a system which does not fall under Brockett's Theorem and thus has an equilibrium point solution. The method has been found to give optimal solutions for all cases of the mobile robot problem. However, in the case where two of the three state boundary conditions are exactly the same at the initial and final time, the optimal solution obtained is a trivial one. The optimality of the solution is subject to the definition of the cost function, the weight of the terminal cost, and choice of rotational gain, α. While closed loop controls of mobile robots are obtainable, they are not optimal in any sense. For an application where efficiency is important, the method demonstrated here would be advantageous.

The results obtained opens up a number of areas for further research into this and related problems:

- Refining the angular feedback $g(U_1)$ such that the control is more proportional to U_1 would result in a more smooth control.

- A more refined algorithm for creating the initial guess of states and costates would help ensure convergence and allow more freedom in choosing problem parameters C and α. The initial guesses could be based on known solutions to some commonly used boundary conditions.

- The method described here requires that the final time be fixed. A more general solution to the free end time, or minimum time problem is desirable.

- This method could be used for in line path planning of a mobile robot. By using the last solution as the new initial guess, the program could constantly update the path for the robot to follow. As the last solution would be a very good guess, the program would converge very quickly.

- The mobile robot problem presented is a kinematics problem. A more realistic second order problem would consider kinetic optimization, mass moment of inertia, smooth velocity profiles and initial and final velocities at zero.

As this list suggests, the possibilities for expanding this research are enormous.

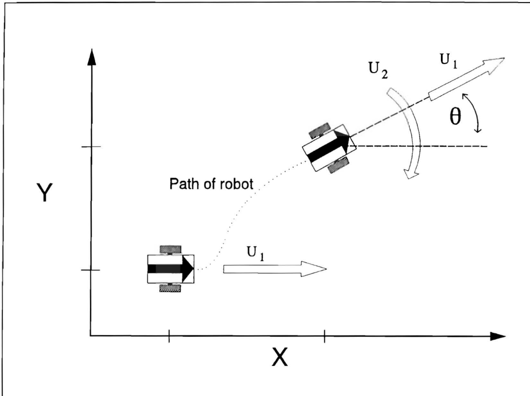


Figure 1

52

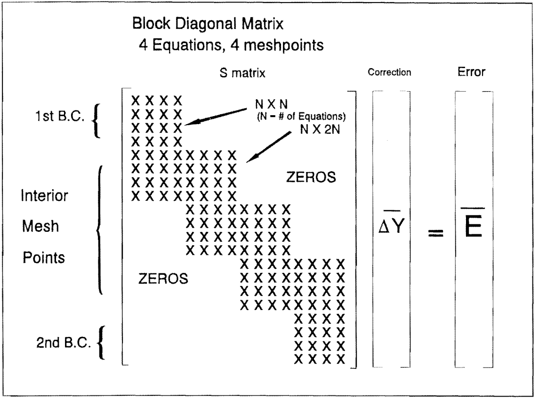

Block Diagonal Matrix
4 Equations, 4 meshpoints
S matrix
Correction
Error
1st B.C.
Interior
Mesh
Points
2nd B.C.
N X N
(N - # of Equations)
N X 2N
ZEROS
ZEROS
$\overline{\Delta Y}$
=
$\overline{E}$

Figure 2

53

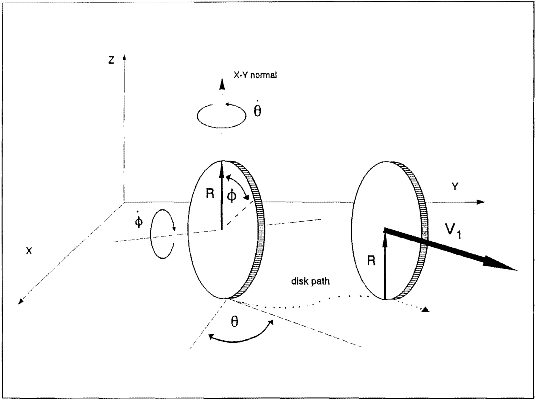


Figure 3

54

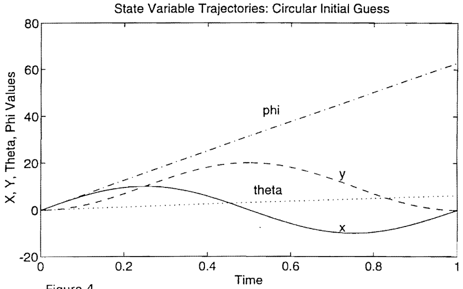


Figure 4

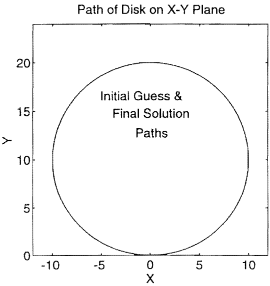


Figure 5

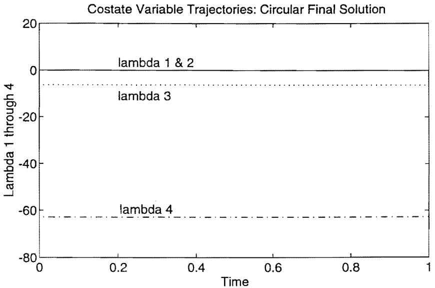


Figure 6

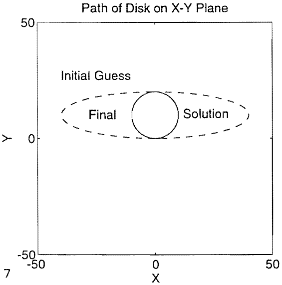


Figure 7

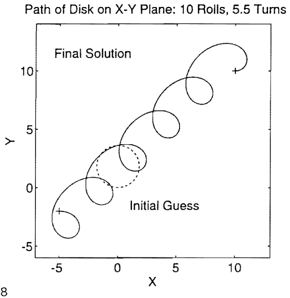


Figure 8

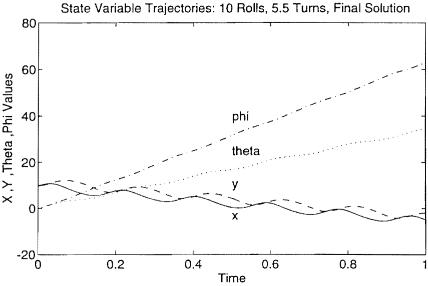


Figure 9

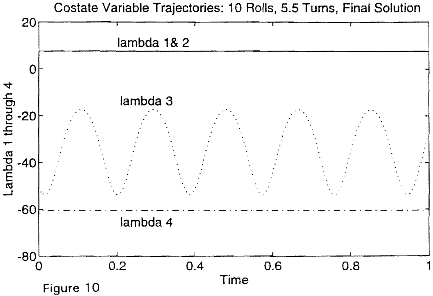


Figure 10

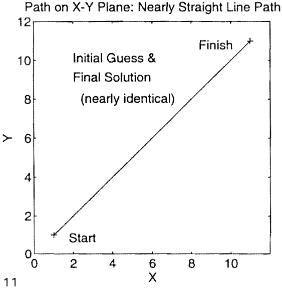


Figure 11

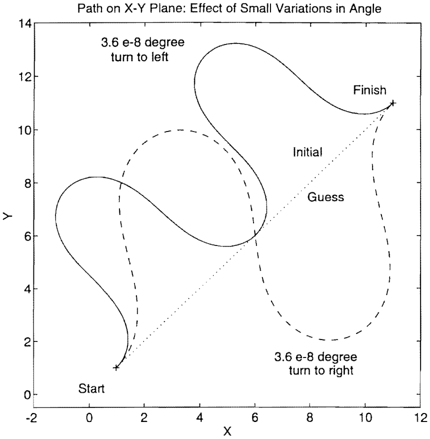
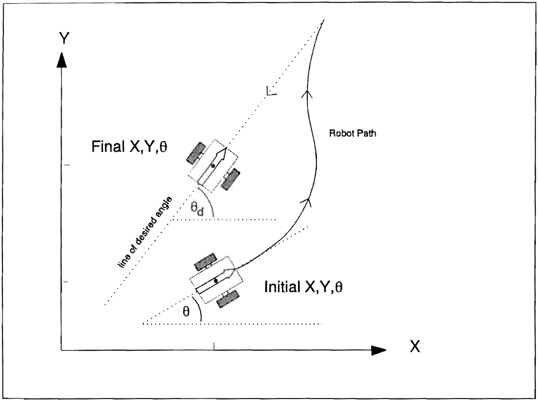

Path on X-Y Plane: Effect of Small Variations in Angle
3.6 e-8 degree
turn to left
Finish
Initial
Guess
3.6 e-8 degree
turn to right
Start
Y
X
14
12
10
8
6
4
2
0
-2
0
2
4
6
8
10
12

Figure 12


Y
X
Final X,Y,θ
Initial X,Y,θ
Robot Path
line of desired angle
θd
θ
Figure 13

60

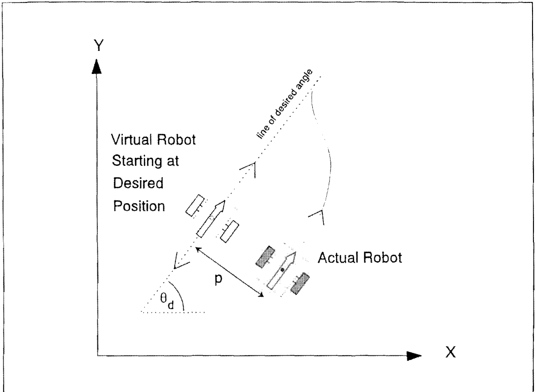

Y
X
line of desired angle
Virtual Robot
Starting at
Desired
Position
Actual Robot
p
θ_d

Figure 14

61

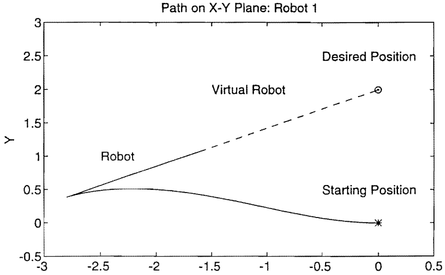


Figure 15

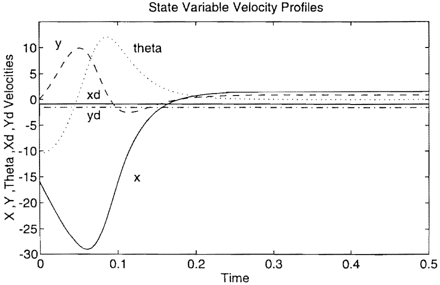


Figure 16

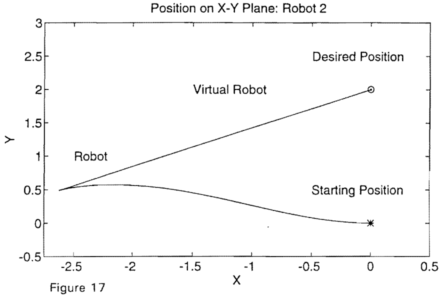


Figure 17

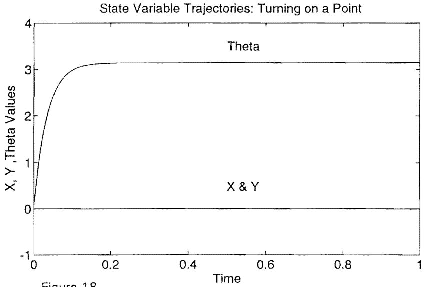


Figure 18

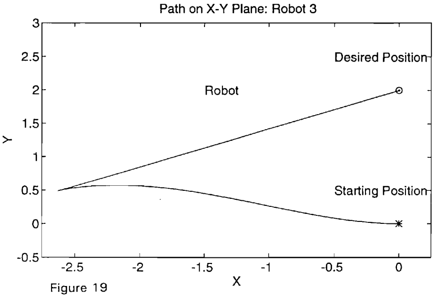


Figure 19

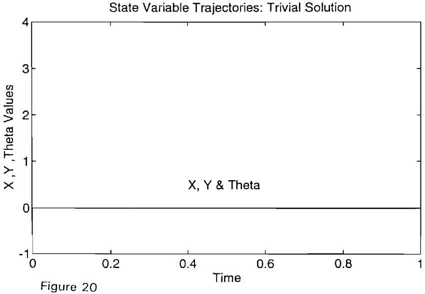


Figure 20

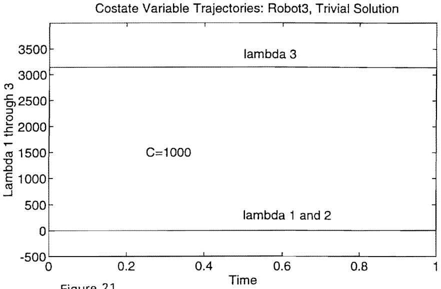


Figure 21

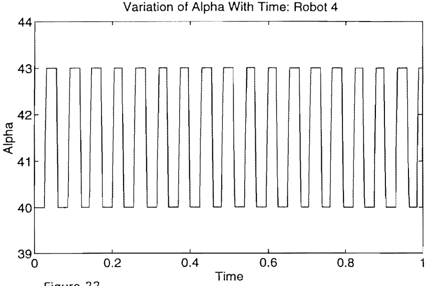


Figure 22

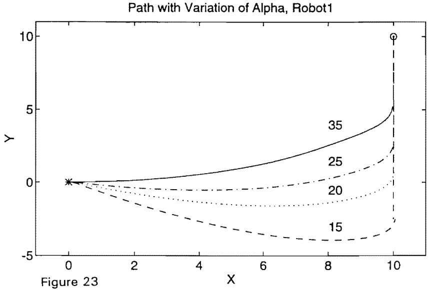


Figure 23

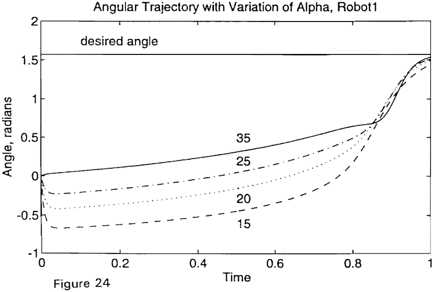


Figure 24

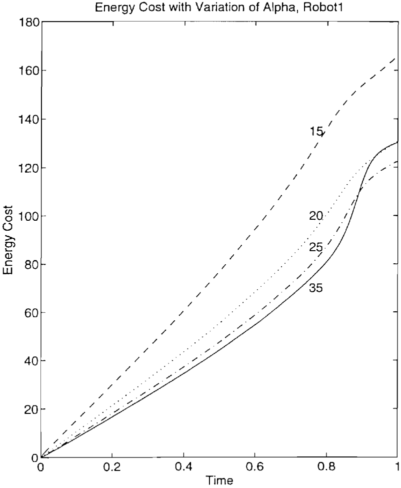

Energy Cost with Variation of Alpha, Robot1
180
160
140
120
100
80
60
40
20
0
Energy Cost
0
0.2
0.4
0.6
0.8
1
Time
15
20
25
35

Figure 25

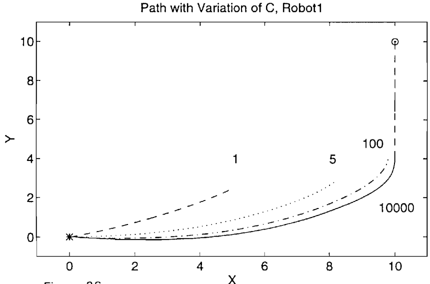


Figure 26

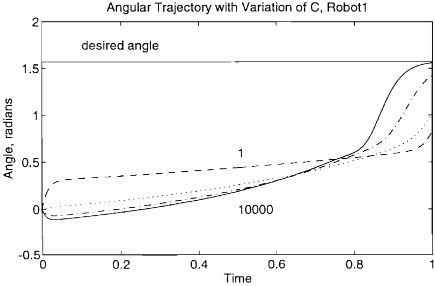
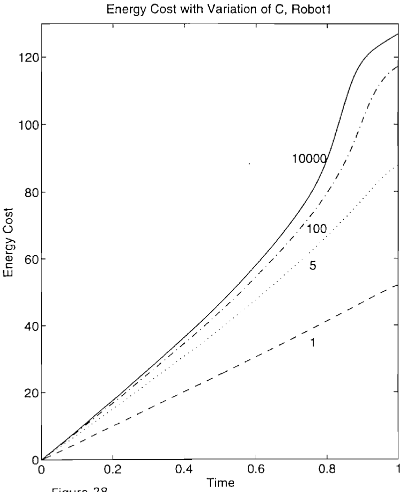


Figure 27


Energy Cost with Variation of C, Robot1
Energy Cost
Time
0
20
40
60
80
100
120
0
0.2
0.4
0.6
0.8
1
10000
100
5
1
Figure 28

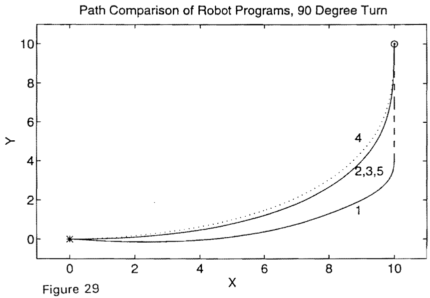


Figure 29

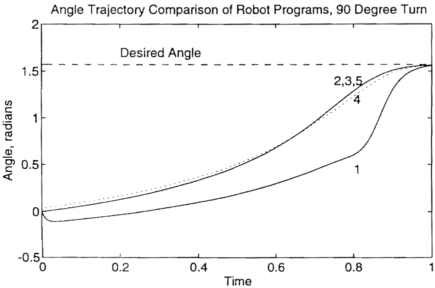


Figure 30

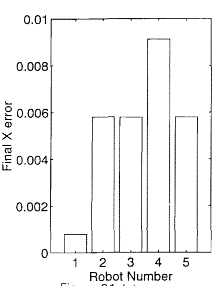


Figure 31 (a)

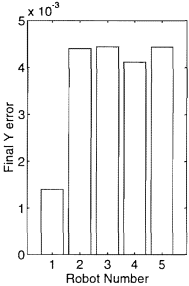


Figure 31 (b)

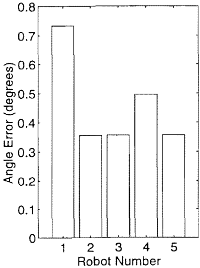


Figure 31 (c)

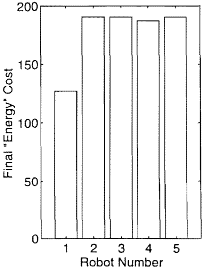
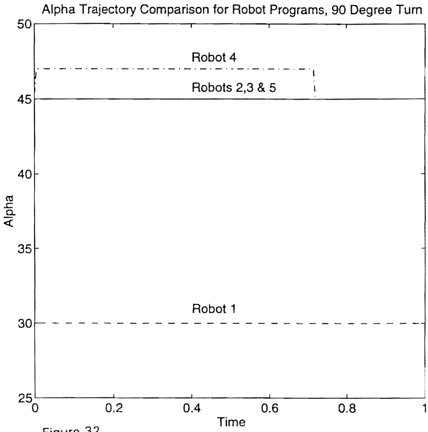


Figure 31 (d)



Alpha Trajectory Comparison for Robot Programs, 90 Degree Turn
Robot 4
Robots 2,3 & 5
Robot 1
Alpha
Time
50
45
40
35
30
25
0
0.2
0.4
0.6
0.8
1

Figure 32

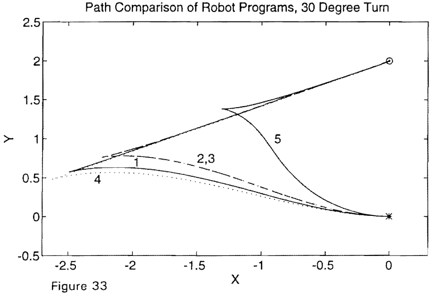


Figure 33

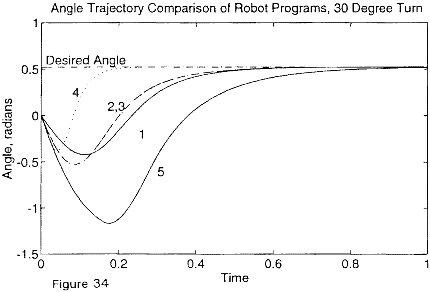


Figure 34

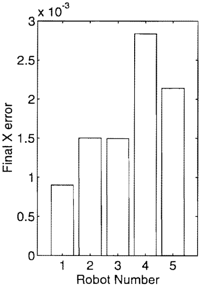


Figure 35 (a)

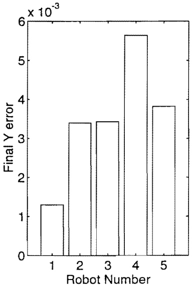


Figure 35 (b)

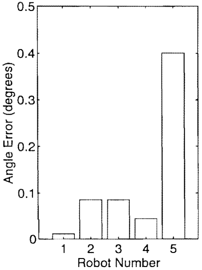


Figure 35 (c)

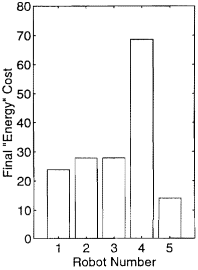


Figure 35 (d)

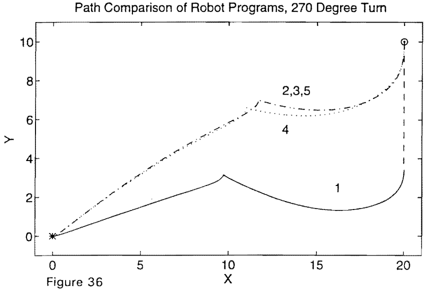


Figure 36

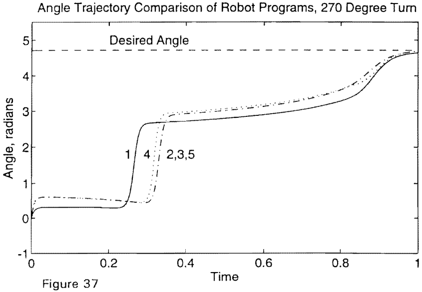


Figure 37

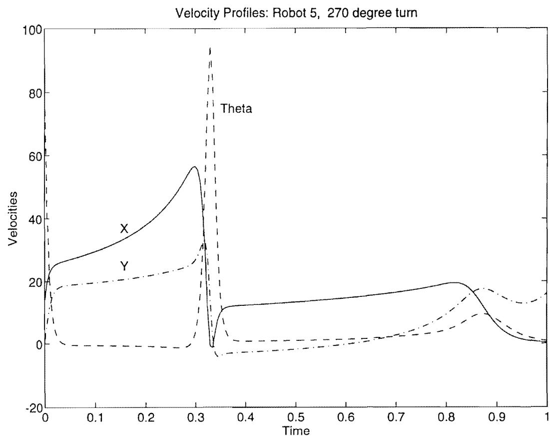

Velocity Profiles: Robot 5, 270 degree turn
Velocities
Time
Theta
X
Y
100
80
60
40
20
0
-20
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Figure 38

76

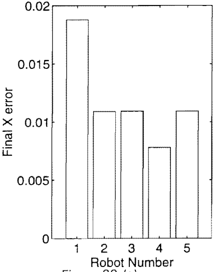


Figure 39 (a)

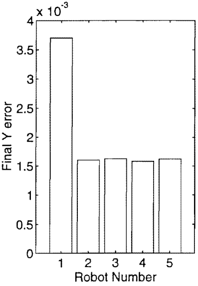


Figure 39 (b)

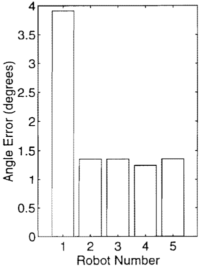


Figure 39 (c)

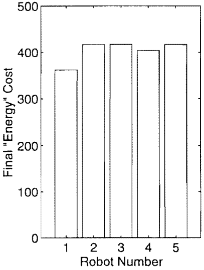


Figure 39 (d)

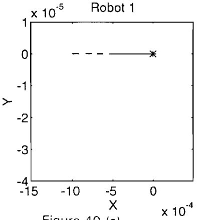


Figure 40 (a)

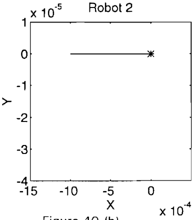


Figure 40 (b)

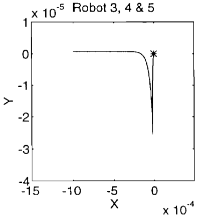


Figure 40 (c)

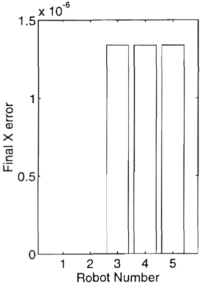


Figure 41 (a)

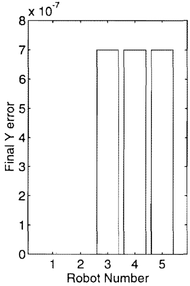


Figure 41 (b)

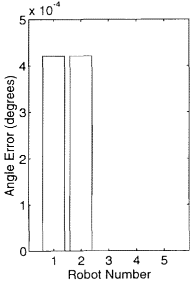


Figure 41 (c)

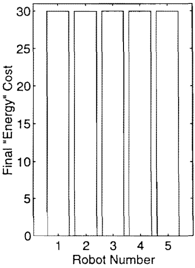


Figure 41 (d)

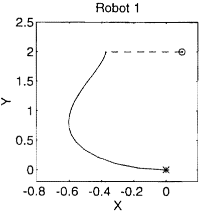


Figure 42 (a)

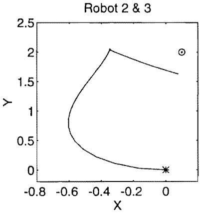


Figure 42 (b)

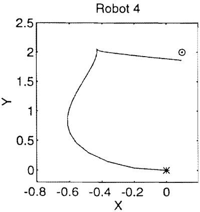


Figure 42 (c)

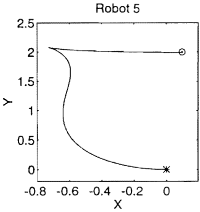


Figure 42 (d)

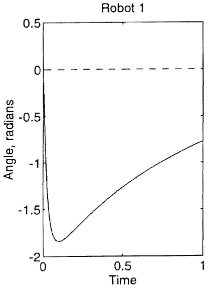


Figure 43 (a)

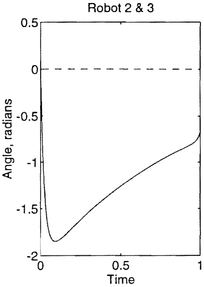


Figure 43 (b)

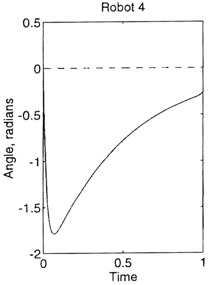


Figure 43 (c)

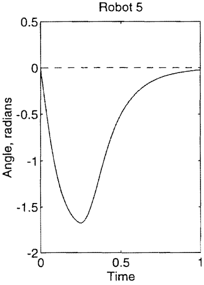


Figure 43 (d)

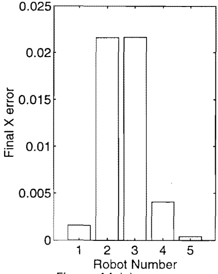


Figure 44 (a)

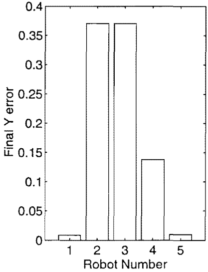


Figure 44 (b)

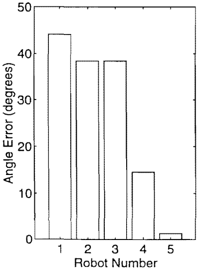


Figure 44 (c)

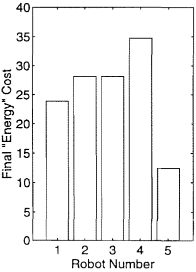


Figure 44 (d)

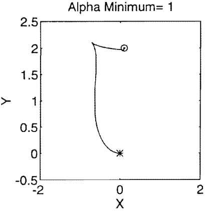


Figure 45 (a)

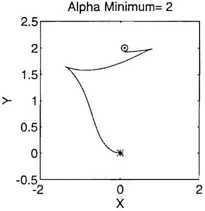


Figure 45 (b)

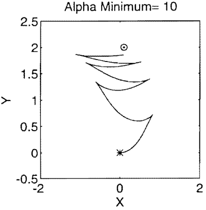


Figure 45 (c)

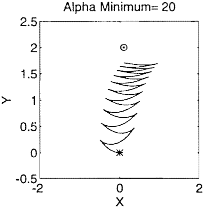


Figure 45 (d)

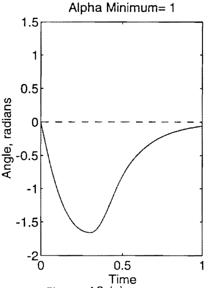


Figure 46 (a)

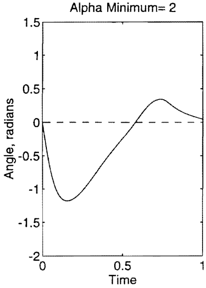


Figure 46 (b)

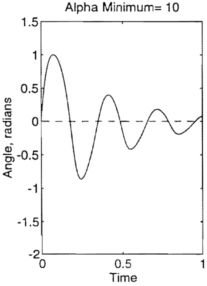


Figure 46 (c)

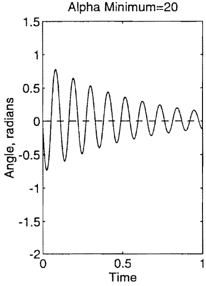


Figure 46 (d)

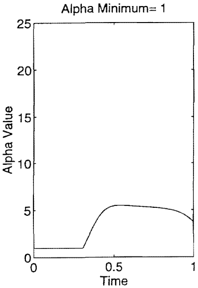


Figure 47 (a)

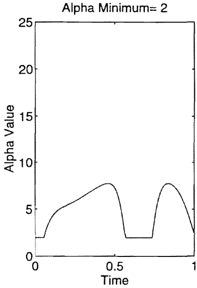


Figure 47 (b)

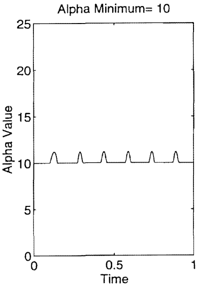


Figure 47 (c)

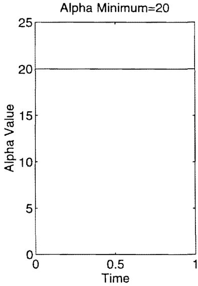


Figure 47 (d)

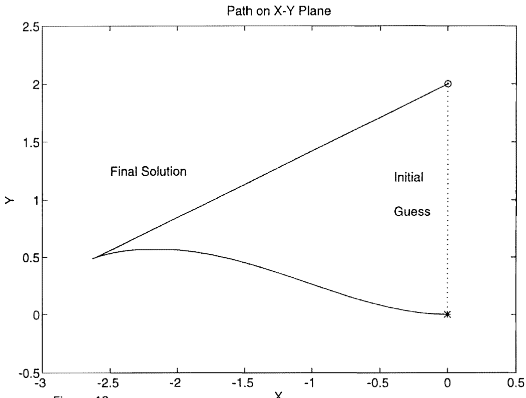

Path on X-Y Plane
Final Solution
Initial
Guess
Y
X
2.5
2
1.5
1
0.5
0
-0.5
-3
-2.5
-2
-1.5
-1
-0.5
0
0.5

Figure 48

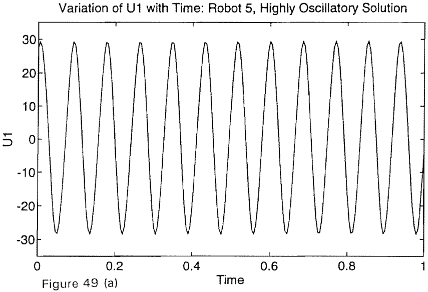


Figure 49 (a)

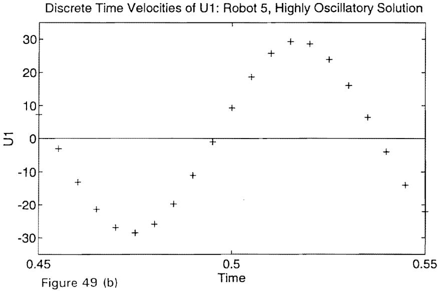


Figure 49 (b)

APPENDIX A

Program Files Specific to Disk

GENDISK.FOR

DIFEQ.FOR

```
      PROGRAM GENDISK
***********************************************************************
*       This is the most general and most interactive form of the disk programs.
*       It works for problems for which a circular initial guess is most
applicable.
*        In some cases the alternate straight line guess may be more applicable.
*       There is an apparent singularity for the exact straight line case.
*       Fortunately this case is readily solved by analytically.
***********************************************************************
*     Source for subroutines Red, Pinvs, Solvde, and Bksub and model for
*     Difeq and Diskmain:
*          Numerical Recipes, William H. Press, et al
***********************************************************************

      IMPLICIT REAL*8 (A-H, O-Z)

      PARAMETER(NE=8,M=201,NB=4,NCI=NE,NCJ=NE-NB+1,NCK=M+1,NSI=NE,
     *     NSJ=2*NE+1, NYJ=NE, NYK=M)
***********************************************************************
*     Variable description:
*
*     GENERAL PROGRAM VARIABLES:
*-----------------------------------------------------------------------
*     NE:    Number of independent equations describing system
*     M:     Number of Meshpoints, divisions of independent variable, time
*     NB:    Number of Boundary Conditions known at initial condition
*     C:     3-D Array for storage of corrections for each iteration
*                Note: largest array in program
*     NCI, NCJ, NCK:
*            dimension variables of C array, must satisfy equations
*            found in parameter statement
*     S:     array for storage of blocks of solution of Difeq.
*     NSI, NSJ:
*            dimension variables of 2-D S array, must satisfy equations
*            found in parameter statement
*     Y:     2-D array containing initial guess for each point.  This array
*            is updated by calculated corrections.  When the corrections
*            are sufficiently small, convergence is acheived.
*     X:     Array for independent variable, time. Used only for comparison
*            of dependent variables after program completes.
*     SCALV: Array of values representing the typical magnitude of the
*            dependent variables.  Used for controlling correction magnitude.
*     INDEXV:lists column ordering for variables in S array, not used in this
*            program
*     ITMAX: Maximum number of iterations
*     CONV:  Convergence criteria for corrections to Y
*     SLOWC: Controls fraction of corrections applied to Y
*         H: Increment of independent variable, divisions between mesh points
*
*     PROBLEM SPECIFIC VARIABLES:
*-----------------------------------------------------------------------
*     RAD:   Radius of Disk
*     ROLL:  Number of revolutions the disk is required to make
*     TURNS: Number of turns the disk is required to make
***********************************************************************

      DIMENSION SCALV(NE),INDEXV(NE),Y(NE,M),C(NCI,NCJ,NCK),S(NSI,NSJ)
      COMMON X(M), H, RAD, ROLL, TURNS, xo,xf,yo,yf,to,tf,po,pf

      OPEN(21,FILE='xplot', STATUS='UNKNOWN')
      OPEN(22,FILE='lplot', STATUS='UNKNOWN')
      OPEN(30,FILE='xiplot', STATUS='UNKNOWN')
      OPEN(31,FILE='itplot', STATUS='UNKNOWN')
      OPEN(32,FILE='testdat', STATUS='UNKNOWN')
      OPEN(33,FILE='liplot', STATUS='UNKNOWN')

      H=1./(M-1)
      PI= 3.141592654
```

```
      Print *,'Enter Test Number.'
      Read *, ITESTNO
      Print *,'Enter the maximum number of iterations.'
      Read *, ITMAX
      Print *,'Enter the convergence criteria.'
      Read *, CONV
      Print *,'Enter SLOWC.'
      Read *, SLOWC
      Print *,'Enter disk radius.'
      Read *, RAD
      Print *,'Enter 1 for circular initial guess,'
      Print *,'Enter 2 for straight line initial guess,'
      Print *,'Enter 3 for an elliptical initial guess,'
      Read *, IGUESS

      IF(IGUESS.EQ.3)THEN
         PRINT *, 'Enter factor for width of elipse'
      Read *, A
         PRINT *, 'Enter factor for height of elipse'
      Read *, B
      ENDIF

C Boundary Conditions:

      Print *,'Enter the starting X-Y coordinates of the disk.'
      Read *, xo,yo
      Print *,'Enter the final X-Y coordinates of the disk.'
      Read *, xf,yf
      Print *,'Enter starting theta in degrees.'
      Read *, to
      to = to*PI/180
      Print *,'Enter starting phi in degrees.'
      Read *, po
      po = po*PI/180
      Print *,'Enter the number of required rolls of the disk.'
      Read *, ROLL
      pf= po + 2*PI*RAD*ROLL
       Print *,'Enter the number of required turns of the disk.'
      Read *, TURNS
      tf= to + 2*PI*TURNS

      Write (32,*) 'ITESTNO =',ITESTNO
      Write (32,*) 'ITMAX =',ITMAX
      Write (32,*) 'CONV =',CONV
      Write (32,*) 'SLOWC =',SLOWC
      Write (32,*) 'RAD =',RAD
      Write (32,*) 'ROLL =',ROLL
      Write (32,*) 'TURNS =',TURNS

       Write (32,*) 'xo =',xo
      Write (32,*) 'yo =',yo
      Write (32,*) 'theta 0 =',to
      Write (32,*) 'phi 0 =',po
       Write (32,*) 'xf =',xf
      Write (32,*) 'yf =',yf
      Write (32,*) 'theta f =',tf
      Write (32,*) 'phi f =',pf

C     NO INDEX CHANGES NECESSARY

      INDEXV(1)=1
      INDEXV(2)=2
      INDEXV(3)=3
      INDEXV(4)=4
      INDEXV(5)=5
      INDEXV(6)=6
      INDEXV(7)=7
      INDEXV(8)=8
```

```
C      Initialize independent vector X (time)

       DO 11 K=1,M
           X(K)=(K-1)*H
 11    CONTINUE

C      Enter initial guess values for all meshpoints

         IF (IGUESS.EQ.1) THEN

C      CIRCULAR INITIAL GUESS WITH LAMBDA'S NEAR KNOWN CIRCLE SOLUTION:
       DO 12 K=1,M
           y(3,K)=(K-1)*2*PI*TURNS/(M-1)
           y(1,K)=ROLL*RAD*SIN(y(3,K))/TURNS
           y(2,K)=ROLL*RAD*(1-COS(y(3,K)))/TURNS
           y(4,K)=(K-1)*2*PI*ROLL*RAD/(M-1)
           y(5,K)=0
           y(6,K)=0
           y(7,K)=-2*PI
           y(8,K)=-2*PI*ROLL*RAD
 12    CONTINUE

C      Scalv set to approximate size of typical values for circle

       SCALV(1)=ROLL*RAD/2
       SCALV(2)=ROLL*RAD
       SCALV(3)=PI*TURNS
       SCALV(4)=ROLL*PI
       SCALV(5)=.05
       SCALV(6)=.05
       SCALV(7)=1
       SCALV(8)=10

         ELSEIF(IGUESS.EQ.2) THEN

C      STRAIGHT LINE INITIAL GUESS:
       DO 14 K=1,M
           y(1,K)=(xf-xo)*(K-1)/(M-1)+xo
           y(2,K)=(yf-yo)*(K-1)/(M-1)+yo
           y(3,K)=ATAN((xf-xo)/(yf-yo))
           y(4,K)=(K-1)*2*PI*ROLL/(M-1)
           y(5,K)=7.77
           y(6,K)=7.77
           y(7,K)=0.001
           y(8,K)=-25.13238
 14    CONTINUE

C      Scalv set to approximate size of typical values for LINE

       SCALV(1)=1
       SCALV(2)=ROLL*RAD*PI
       SCALV(3)=1
       SCALV(4)=ROLL*RAD*PI
       SCALV(5)=1
       SCALV(6)=1
       SCALV(7)=1
       SCALV(8)=1

         ELSEIF (IGUESS.EQ.3) THEN

C      ELLIPTICAL INITIAL GUESS WITH LAMBDA'S NEAR KNOWN CIRCLE SOLUTION:
       DO 15 K=1,M
           y(3,K)=(K-1)*2*PI*TURNS/(M-1)
           y(1,K)=A*ROLL*RAD*SIN(y(3,K))/TURNS
           y(2,K)=B*ROLL*RAD*(1-COS(y(3,K)))/TURNS
           y(4,K)=(K-1)*2*PI*ROLL*RAD/(M-1)
           y(5,K)=0
           y(6,K)=0
           y(7,K)=-2*PI
```

```
        y(8,K)=-2*PI*ROLL*RAD
 15   CONTINUE

C     Scalv set to approximate size of typical values for circle

      SCALV(1)=ROLL*RAD/2
      SCALV(2)=ROLL*RAD
      SCALV(3)=PI*TURNS
      SCALV(4)=ROLL*PI
      SCALV(5)=.05
      SCALV(6)=.05
      SCALV(7)=1
      SCALV(8)=10

        ENDIF

C     Write initial guess to file

      DO 13 K=1,M
           WRITE(30,80) X(K),Y(1,K),Y(2,K),Y(3,K),Y(4,K)
           WRITE(33,80) X(K),Y(5,K),Y(6,K),Y(7,K),Y(8,K)
 13   CONTINUE

 80   FORMAT(2X,5F10.5)


C     EXPLICIT ENTRY OF BOUNDARY CONDITIONS:

        y(1,1)=xo
        y(1,M)=xf
        y(2,1)=yo
        y(2,M)=yf
        y(3,1)=to
        y(3,M)=tf
        y(4,1)=po
        y(4,M)=pf


      CALL SOLVDE(ITMAX,CONV,SLOWC,SCALV,INDEXV,NE,NB,M,Y,NYJ,NYK,
     *     C,NCI,NCJ,NCK,S,NSI,NSJ)

C     Write final Y values to file:

         DO 181 k = 1,201
           WRITE(21,83) X(K),Y(1,k),Y(2,k),Y(3,k),Y(4,k)
           WRITE(22,83) X(K),Y(5,k),Y(6,k),Y(7,k),Y(8,k)
 181     CONTINUE

 83      FORMAT(2x,5F15.5)

        PRINT *,'PROGRAM COMPLETED'

       CLOSE(21)
       CLOSE(22)
       CLOSE(30)
       CLOSE(31)
       CLOSE(32)
       CLOSE(33)

       END
```

```
      SUBROUTINE DIFEQ(K,K1,K2,JSF,IS1,ISF,INDEXV,NE,S,NSI,NSJ,Y,NYJ,
     *                  NYK)
      IMPLICIT REAL*8 (A-H, O-Z)

      PARAMETER(M=201)
      DIMENSION Y(NYJ,NYK), S(NSI,NSJ), INDEXV(NYJ)
      COMMON X(M), H, RAD, ROLL, TURNS, xo,xf,yo,yf,to,tf,po,pf

ccc      PI = DACOS(-1)
      PI = 3.14159

C     Initialize matrix S as 0

      DO 10 I=1,NSI
          DO 9 J=1,NSJ
              S(I,J) = 0.0
 9        CONTINUE
 10   CONTINUE

*******************************************************************
C     Initial Boundary Conditions

      IF(K.EQ.K1) THEN

C     Enter non-zero values:

          DO 11 I= 1,4
              S(4+I,8+I)=1.0
 11       CONTINUE

C    Initial values in right hand vector for initial block

          S(5,JSF)= y(1,1)-xo
          S(6,JSF)= y(2,1)-yo
          S(7,JSF)= y(3,1)-to
          S(8,JSF)= y(4,1)-po

*******************************************************************
C     End Boundary Conditions

      ELSE IF (K.GT.K2) THEN

C     Enter non-zero values:

          DO 12 I= 1,4
              S(I,8+I)=1.0
 12       CONTINUE

C    Final values in right hand vector for final block

          S(1,JSF)= y(1,M)-xf
          S(2,JSF)= y(2,M)-yf
          S(3,JSF)= y(3,M)-tf
          S(4,JSF)= y(4,M)-pf

*******************************************************************
C     Interior Points
C         Derived from Finite Difference Equations of Motion

      ELSE

C     Pre-calculation of commonly used variables:

      Y3=Y(3,K)+Y(3,K-1)
      Y5=Y(5,K)+Y(5,K-1)
      Y6=Y(6,K)+Y(6,K-1)
      Y7=Y(7,K)+Y(7,K-1)
      Y8=Y(8,K)+Y(8,K-1)
```

```
      r1 = 1
      r2 = 1

c     Enter non-zero values:

      S(1,1) = -1
      S(1,2) = 0
      S(1,3) = h*Rad**2*Y6*Cos(Y3)/(4*r2)-h*Rad*Y8*Sin(Y3/2)/(4*r2)-
     & h*Rad**2*Y5*Sin(Y3)/(4*r2)
      S(1,4) = 0
      S(1,5) = h*Rad**2/(4*r2) + h*Rad**2*Cos(Y3)/(4*r2)
      S(1,6) = h*Rad**2*Sin(Y3)/(4*r2)
      S(1,7) = 0
      S(1,8) = h*Rad*Cos(Y3/2)/(2*r2)
      S(1,9) = 1
      S(1,10) = 0
      S(1,11) = s(1,3)
      S(1,12) = 0
      S(1,13) = s(1,5)
      S(1,14) = s(1,6)
      S(1,15) = 0
      S(1,16) = s(1,8)
      S(2,1) = 0
      S(2,2) = -1
      S(2,3) = h*Rad*Y8*Cos(Y3/2)/(4*r2)+h*Rad**2*Y5*Cos(Y3)/(4*r2)+
     & h*Rad**2*Y6*Sin(Y3)/(4*r2)
      S(2,4) = 0
      S(2,5) = h*Rad**2*Sin(Y3)/(4*r2)
      S(2,6) = h*Rad**2/(4*r2) - h*Rad**2*Cos(Y3)/(4*r2)
      S(2,7) = 0
      S(2,8) = h*Rad*Sin(Y3/2)/(2*r2)
      S(2,9) = 0
      S(2,10) = 1
      S(2,11) = s(2,3)
      S(2,12) = 0
      S(2,13) = s(1,6)
      S(2,14) = s(2,6)
      S(2,15) = 0
      S(2,16) = s(2,8)
      S(3,1) = 0
      S(3,2) = 0
      S(3,3) = -1
      S(3,4) = 0
      S(3,5) = 0
      S(3,6) = 0
      S(3,7) = h/(2*r1)
      S(3,8) = 0
      S(3,9) = 0
      S(3,10) = 0
      S(3,11) = 1
      S(3,12) = 0
      S(3,13) = 0
      S(3,14) = 0
      S(3,15) = s(3,7)
      S(3,16) = 0
      S(4,1) = 0
      S(4,2) = 0
      S(4,3) = h*Rad*Y6*Cos(Y3/2)/(4*r2) - h*Rad*Y5*Sin(Y3/2)/(4*r2)
      S(4,4) = -1
      S(4,5) = h*Rad*Cos(Y3/2)/(2*r2)
      S(4,6) = h*Rad*Sin(Y3/2)/(2*r2)
      S(4,7) = 0
      S(4,8) = h/(2*r2)
      S(4,9) = 0
      S(4,10) = 0
      S(4,11) = s(4,3)
      S(4,12) = 1
      S(4,13) = s(1,8)
      S(4,14) = s(2,8)
```

```
 S(4,15) = 0
 S(4,16) = s(4,8)
 S(5,1) = 0
 S(5,2) = 0
 S(5,3) = 0
 S(5,4) = 0
 S(5,5) = -1
 S(5,6) = 0
 S(5,7) = 0
 S(5,8) = 0
 S(5,9) = 0
 S(5,10) = 0
 S(5,11) = 0
 S(5,12) = 0
 S(5,13) = 1
 S(5,14) = 0
 S(5,15) = 0
 S(5,16) = 0
 S(6,1) = 0
 S(6,2) = 0
 S(6,3) = 0
 S(6,4) = 0
 S(6,5) = 0
 S(6,6) = -1
 S(6,7) = 0
 S(6,8) = 0
 S(6,9) = 0
 S(6,10) = 0
 S(6,11) = 0
 S(6,12) = 0
 S(6,13) = 0
 S(6,14) = 1
 S(6,15) = 0
 S(6,16) = 0
 S(7,1) = 0
 S(7,2) = 0
 S(7,3) = h*Rad*Y5*Y8*Cos(Y3/2)/(8*r2)+
&  h*Rad**2*Y5**2*Cos(Y3)/(8*r2) -
&  h*Rad**2*Y6**2*Cos(Y3)/(8*r2) +
&  h*Rad*Y6*Y8*Sin(Y3/2)/(8*r2) +
&  h*Rad**2*Y5*Y6*Sin(Y3)/(4*r2)
 S(7,4) = 0
 S(7,5) = -(h*Rad**2*Y6*Cos(Y3))/(4*r2)+h*Rad*Y8*Sin(Y3/2)/(4*r2)+
&  h*Rad**2*Y5*Sin(Y3)/(4*r2)
 S(7,6) = -(h*Rad*Y8*Cos(Y3/2))/(4*r2)-h*Rad**2*Y5*Cos(Y3)/(4*r2)-
&  h*Rad**2*Y6*Sin(Y3)/(4*r2)
 S(7,7) = -1
 S(7,8) = -(h*Rad*Y6*Cos(Y3/2))/(4*r2) + h*Rad*Y5*Sin(Y3/2)/(4*r2)
 S(7,9) = 0
 S(7,10) = 0
 S(7,11) = s(7,3)
 S(7,12) = 0
 S(7,13) = s(7,5)
 S(7,14) = s(7,6)
 S(7,15) = 1
 S(7,16) = s(7,8)
 S(8,1) = 0
 S(8,2) = 0
 S(8,3) = 0
 S(8,4) = 0
 S(8,5) = 0
 S(8,6) = 0
 S(8,7) = 0
 S(8,8) = -1
 S(8,9) = 0
 S(8,10) = 0
 S(8,11) = 0
 S(8,12) = 0
 S(8,13) = 0
```

```
      S(8,14) = 0
      S(8,15) = 0
      S(8,16) = 1

      S(1,JSF) = h*Rad**2*Y5/(4*r2) + h*Rad*Y8*Cos(Y3/2)/(2*r2) +
     &  h*Rad**2*Y5*Cos(Y3)/(4*r2) +
     &  h*Rad**2*Y6*Sin(Y3)/(4*r2) - Y(1,-1 + K) + Y(1,K)
      S(2,JSF) = h*Rad**2*Y6/(4*r2) - h*Rad**2*Y6*Cos(Y3)/(4*r2) +
     &  h*Rad*Y8*Sin(Y3/2)/(2*r2) + h*Rad**2*Y5*Sin(Y3)/(4*r2) -
     &  Y(2,-1 + K) + Y(2,K)
      S(3,JSF) = h*Y7/(2*r1) - Y(3,-1 + K) + Y(3,K)
      S(4,JSF) = h*Y8/(2*r2) + h*Rad*Y5*Cos(Y3/2)/(2*r2) +
     &  h*Rad*Y6*Sin(Y3/2)/(2*r2) - Y(4,-1 + K) + Y(4,K)
      S(5,JSF) = -Y(5,-1 + K) + Y(5,K)
      S(6,JSF) = -Y(6,-1 + K) + Y(6,K)
      S(7,JSF) = -(h*Rad*Y6*Y8*Cos(Y3/2))/(4*r2) -
     &  h*Rad**2*Y5*Y6*Cos(Y3)/(4*r2) +
     &  h*Rad*Y5*Y8*Sin(Y3/2)/(4*r2) +
     &  h*Rad**2*Y5**2*Sin(Y3)/(8*r2) -
     &  h*Rad**2*Y6**2*Sin(Y3)/(8*r2) - Y(7,-1 + K) + Y(7,K)
      S(8,JSF) = -Y(8,-1 + K) + Y(8,K)

           ENDIF


C     Dummy use of variables to prevent inocculous warning on MS Compiler
C     (Variables not used)
      IS1 = IS1
      ISF = ISF
      INDEXV(1) = INDEXV(1)
      NE = NE

      RETURN
      END
```

APPENDIX B

Program Files Specific to Robot 1

ROBOT1.FOR

DIFEQ.FOR (for Robot 1)

XLATOR.FOR

```
      PROGRAM ROBOT1
**********************************************************************
*     Source for subroutines Red, Pinvs, Solvde, and Bksub and model for
*     Difeq and Diskmain:
*          Numerical Recipes, William H. Press, et al
**********************************************************************

      IMPLICIT REAL*8 (A-H, O-Z)

      PARAMETER(NE=6,M=201,NB=3,NCI=NE,NCJ=NE-NB+1,NCK=M+1,NSI=NE,
     &     NSJ=2*NE+1, NYJ=NE, NYK=M)

      DIMENSION SCALV(NE),INDEXV(NE),Y(NE,M),C(NCI,NCJ,NCK),S(NSI,NSJ),
     &          YDOT(NE-1,M),POS(NE-1,M)
      COMMON X(M), H, DELX0, DELY0, DELTHETO, THETAD, EPS, ALPHA, CWT

**********************************************************************
*     Variable description:
*
*     GENERAL PROGRAM VARIABLES:
*---------------------------------------------------------------------
*     NE:     Number of independent equations describing system
*     M:      Number of Meshpoints, divisions of independent variable, time
*     NB:     Number of Boundary Conditions known at initial condition
*     C:      3-D Array for storage of corrections for each iteration
*                Note: largest array in program
*     NCI, NCJ, NCK:
*             dimension variables of C array, must satisfy equations
*             found in parameter statement
*     S:      array for storage of blocks of solution of Difeq.
*     NSI, NSJ:
*             dimension variables of 2-D S array, must satisfy equations
*             found in parameter statement
*     Y:      2-D array containing initial guess for each point.  This array
*             is updated by calculated corrections.  When the corrections
*             are sufficiently small, convergence is acheived.
*     X:      Array for independent variable, time. Used only for comparison
*             of dependent variables after program completes.
*     SCALV: Array of values representing the typical magnitude of the
*             dependent variables.  Used for controlling correction magnitude.
*     INDEXV:lists column ordering for variables in S array, not used in this
*             program
*     ITMAX: Maximum number of iterations
*     CONV:  Convergence criteria for corrections to Y
*     SLOWC: Controls fraction of corrections applied to Y
*         H: Increment of independent variable, divisions between mesh points
*
*     PROBLEM SPECIFIC VARIABLES:
*---------------------------------------------------------------------
*
*       X0:                Initial X coordinate of robot
*       Y0:                Initial Y coordinate of robot
*       THETA0:                 Initial angle wrt X axis of robot
*       XF:                Desired final X coordinate of robot
*       YF:                Desired final Y coordinate of robot
*       THETAD:                 Desired final angle coordinate of robot
*       DELX0:             Initial boundary condition for state variable delta-X
*       DELY0:             Initial boundary condition for state variable delta-Y
*       DELTHETO:          Initial boundary condition for state variable delta-theta
*       EPS:               Smallest value for which f(delta-x)=Sin(delta-x)/(delta-x)
*       EPS2:              DUMMY VARIABLE USED FOR CONTINUITY WITH OTHER FORMS OF
ROBOT.
*       ALPHA:             Rotational gain related to delta-theta-dot
*       ALPHAMAX:          DUMMY VARIABLE USED FOR CONTINUITY WITH OTHER FORMS OF
ROBOT.
*       CWT:               Weighting parameter for Terminal Costs
*       RL1,RL2,RL3:       Initial guess amplitude for lambda costates.
*       OHM1,OHM2,OHM3:         Initial guess frequencies for lambda costates.
*       DC1,DC2,DC3:       Initial guess d.c. offsets for lambda costates.
```

```
*       PHI1,PHI2,PHI3:        Initial guess phase lag for lambda costates.
*       SL1,SL2,SL3:   Initial guess SCALV, scale sizes, for lambda costates.
*       POS(NE-1,M):   Position Trajectory for x,y,theta, xd and yd.
*
**********************************************************************

      OPEN(21,FILE='xplot.rob1', STATUS='UNKNOWN')
      OPEN(22,FILE='lplot.rob1', STATUS='UNKNOWN')
      OPEN(30,FILE='xiplot.rob1', STATUS='UNKNOWN')
      OPEN(31,FILE='liplot.rob1', STATUS='UNKNOWN')
      OPEN(32,FILE='itplot.rob1', STATUS='UNKNOWN')
      OPEN(33,FILE='testdat.rob1', STATUS='UNKNOWN')
      OPEN(34,FILE='dotplot.rob1', STATUS='UNKNOWN')
      OPEN(35,FILE='posplot.rob1', STATUS='UNKNOWN')
      OPEN(37,FILE='ulplot.rob1', STATUS='UNKNOWN')

      PI= 3.141592654

      PRINT *,'ENTER ITMAX, CONV, SLOWC'
      READ *, ITMAX, CONV, SLOWC
      PRINT *,'ENTER ALPHA'
      READ *, ALPHA
       print *,'DUMMY READ FOR DATA FILE COMPATABILITY, Enter #'
       READ *, DUMMY
      PRINT *,'ENTER C, WEIGHTING PARAMTER FOR TERMINAL COST'
      READ *, CWT
      PRINT *,'ENTER EPSILON'
      READ *, EPS
       print *,'DUMMY READ FOR DATA FILE COMPATABILITY, Enter #'
       READ *, DUMMY
      PRINT *, 'ENTER INITIAL X,Y, AND THETA(degrees)'
      READ *, X0,Y0,THETA0
      THETA0=THETA0*PI/180
      PRINT *, 'ENTER FINAL X,Y, AND THETA(degrees)'
      READ *, XF,YF,THETAD
      THETAD=THETAD*PI/180
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA AMPLITUDES'
      READ *, RL1,RL2,RL3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA FREQUENCIES'
      READ *, OHM1,OHM2,OHM3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA DC OFFSETS'
      READ *, DC1,DC2,DC3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA PHASES'
      READ *, PHI1, PHI2, PHI3
       PHI1=PHI1*PI/180
       PHI2=PHI2*PI/180
       PHI3=PHI3*PI/180
      PRINT *,'ENTER INITIAL GUESS FOR SIZE OF 3 LAMBDA VALUES'
      READ *, SL1,SL2,SL3

      DELX0=(XF-X0)
      DELY0=(YF-Y0)
      DELTHET0=(THETAD-THETA0)

      H=1./(M-1)

C     NO INDEX CHANGES NECESSARY
C     Index Scale used by SOLVDE.FOR

      INDEXV(1)=1
      INDEXV(2)=2
      INDEXV(3)=3
      INDEXV(4)=4
      INDEXV(5)=5
      INDEXV(6)=6

c     INITIAL GUESS FOR ALL POINTS, 1 - M

C     Initialize independent vector X (time)
```

```
      DO 11 K=1,M
          X(K)=(K-1)*H
 11   CONTINUE

C     Enter initial values for all meshpoints
C       NOTE: BOUNDARY CONDITONS FOR Y(1)-Y(3) ARE
C             ENTERED AT POINTS 1 AND M DURING THE INITIAL GUESS!!!
C             THESE NUMBERS MUST COINCIDE WITH ANY DESIRED B.C.!!!

C      INITIAL GUESS:
      DO 12 K=1,M
          y(1,K)=DELX0-(DELX0*(K-1)/(M-1))
          y(2,K)=DELY0-(DELY0*(K-1)/(M-1))
          y(3,K)=DELTHETO-(DELTHETO*(K-1)/(M-1))
          y(4,K)=RL1*SIN(2*PI*OHM1*(K-1)/(M-1)+PHI1)+DC1
          y(5,K)=RL2*SIN(2*PI*OHM2*(K-1)/(M-1)+PHI2)+DC2
          y(6,K)=RL3*SIN(2*PI*OHM3*(K-1)/(M-1)+PHI3)+DC3
 12   CONTINUE

C     Write initial guess to file

      DO 13 K=1,M
           WRITE(30,80) X(K),Y(1,K),Y(2,K),Y(3,K)
           WRITE(31,80) X(K),Y(4,K),Y(5,K),Y(6,K)
 13        CONTINUE

C     Scalv set to approximate size of typical values of known solution

      SCALV(1)=ABS(DELX0/2)+.01
      SCALV(2)=ABS(DELY0/2)+.01
      SCALV(3)=ABS(2*DELTHETO/M)+.01
      SCALV(4)=ABS(SL1)+.01
      SCALV(5)=ABS(SL2)+.01
      SCALV(6)=ABS(SL3)+.01

C    WRITE TEST DATA TO FILE
      WRITE(33,*) 'ITMAX =',ITMAX
      WRITE(33,*) 'CONV  =',CONV
      WRITE(33,*) 'SLOWC =',SLOWC
      WRITE(33,*) 'ALPHA =',ALPHA
      WRITE(33,*) 'CWT =',CWT
      WRITE(33,*) 'EPSILON =',EPS
      WRITE(33,*) 'INITIAL X,Y & THETA = ', X0,Y0,(THETA0*180/PI)
      WRITE(33,*) 'FINAL   X,Y & THETA = ', XF,YF,(THETAD*180/PI)
      WRITE(33,*) 'INITIAL LAMBDA VALUES', RL1,RL2,RL3
      WRITE(33,*) 'SIZE OF LAMBDA VALUES', SL1,SL2,SL3
      WRITE(33,*) 'DELX0 =', DELX0
      WRITE(33,*) 'DELY0 =', DELY0
      WRITE(33,*) 'DELTHETO =', DELTHETO

      CALL SOLVDE(ITMAX,CONV,SLOWC,SCALV,INDEXV,NE,NB,M,Y,NYJ,NYK,
     *     C,NCI,NCJ,NCK,S,NSI,NSJ)

C     Write final Y values to file:

        DO 181 k = 1,M
          WRITE(21,80) X(K),Y(1,k),Y(2,k),Y(3,k)
          WRITE(22,80) X(K),Y(4,k),Y(5,k),Y(6,k)
 181    CONTINUE

 80   FORMAT(2X,4F15.4)

      CALL XLATOR(Y,YDOT,H,NYJ,NYK,X,THETAD,EPS,ALPHA,
     &                  X0,Y0,THETA0,XF,YF,POS)

       PRINT *,'PROGRAM COMPLETED'

      CLOSE(21)
      CLOSE(22)
```

```
CLOSE(30)
CLOSE(31)
CLOSE(32)
CLOSE(33)
CLOSE(34)
CLOSE(35)
CLOSE(37)

END
```

```
      SUBROUTINE DIFEQ(K,K1,K2,JSF,IS1,ISF,INDEXV,NE,S,NSI,NSJ,Y,NYJ,
     *                   NYK)
      IMPLICIT REAL*8 (A-H, O-Z)

**********************************************************************
* MODIFIED 7/18/94 TO INCLUDE TERMINAL COST AT FINA BOUNDARY CONDITION
*
*
**********************************************************************
      PARAMETER(M=201)
      DIMENSION Y(NYJ,NYK), S(NSI,NSJ), INDEXV(NYJ)
      COMMON X(M), H, DELX0, DELY0, DELTHETO, THETAD, EPS, ALPHA, CWT


C     Initialize matrix S as 0

      DO 10 I=1,NSI
          DO 9 J=1,NSJ
              S(I,J) = 0.0
 9        CONTINUE
 10   CONTINUE

**********************************************************************
C     Initial Boundary Conditions

      IF(K.EQ.K1) THEN

C     Enter non-zero values:

          DO 11 I= 1,3
              S(3+I,6+I)=1.0
 11       CONTINUE

C    Initial values in right hand vector for initial block

          S(4,JSF)= y(1,1)-DELX0
          S(5,JSF)= y(2,1)-DELY0
          S(6,JSF)= y(3,1)-DELTHETO

**********************************************************************
C     End Boundary Conditions

      ELSE IF (K.GT.K2) THEN

C     Enter non-zero values:

          S(1,7)= CWT
          S(1,8)= 0.0
          S(1,9)= 0.0
          S(1,10)= -1.0
          S(1,11)= 0.0
          S(1,12)= 0.0

          S(2,7)= 0.0
          S(2,8)= CWT
          S(2,9)= 0.0
          S(2,10)= 0.0
          S(2,11)= -1.0
          S(2,12)= 0.0

          S(3,7)= 0.0
          S(3,8)= 0.0
          S(3,9)= CWT
          S(3,10)= 0.0
          S(3,11)= 0.0
          S(3,12)= -1.0

C    Final values in right hand vector for final block
```

```
        S(1,JSF)= Y(1,M)*CWT - Y(4,M)
        S(2,JSF)= Y(2,M)*CWT - Y(5,M)
        S(3,JSF)= Y(3,M)*CWT - Y(6,M)

*****************************************************************
C     Interior Points
C        Derived from Finite Difference Equations of Motion

      ELSE

C     Pre-calculation of commonly used variables:

      Y1=(Y(1,K)+Y(1,K-1))/2.0
      Y2=(Y(2,K)+Y(2,K-1))/2.0
      Y3=(Y(3,K)+Y(3,K-1))/2.0
      Y4=(Y(4,K)+Y(4,K-1))/2.0
      Y5=(Y(5,K)+Y(5,K-1))/2.0
      Y6=(Y(6,K)+Y(6,K-1))/2.0

       CTD=COS(THETAD)
       STD=SIN(THETAD)
       CTDY3=COS(THETAD-Y3)
       STDY3=SIN(THETAD-Y3)

       P= Y2*CTD-Y1*STD

       IF(ABS(Y3).GT.EPS)THEN
          FY3=SIN(Y3)/(Y3)
          FPY3=(COS(Y3)/Y3)-(SIN(Y3)/(Y3**2))
          SIG4=FPY3/2.0
           SIG12=(-SIN(Y3)/Y3-2.0*COS(Y3)/(Y3)**2 +
     &           2.0*SIN(Y3)/(Y3)**3)/2.0
       ELSE
          FY3=1.0
          FPY3=0.0
          SIG4=0.0
          SIG12=-1/6.0
       ENDIF

       U1=Y4*CTDY3 + Y5*STDY3 + Y6*P*FY3
       UD=-Y4*CTD-Y5*STD

       SIG1= -Y6*FY3*STD/2.0
       SIG2= Y6*FY3*CTD/2.0
       SIG3= Y4/2.0*STDY3 - Y5/2.0*CTDY3 + Y6*P*SIG4

       SIG5= -CTD/2.0
       SIG6= CTDY3/2.0
       SIG7= -STD/2.0
       SIG8= STDY3/2.0
       SIG9= P*FY3/2.0
       SIG10 = -STD/2.0
       SIG11= CTD/2.0

c     Enter non-zero values:

       S(1,1)= -1 + H*CTDY3*SIG1
       S(1,2)= H*CTDY3*SIG2
        S(1,3)= H*(CTDY3*SIG3 + STDY3*U1/2.0)
       S(1,4)= -H*(CTD*SIG5 - CTDY3*SIG6)
       S(1,5)= -H*(CTD*SIG7 - CTDY3*SIG8)
       S(1,6)= H*CTDY3*SIG9
       S(1,7)= S(1,1) + 2.0
       S(1,8)= S(1,2)
       S(1,9)= S(1,3)
       S(1,10)= S(1,4)
       S(1,11)= S(1,5)
       S(1,12)= S(1,6)
```

```
      S(2,1)= H*STDY3*SIG1
      S(2,2)= -1 + H*STDY3*SIG2
      S(2,3)= H*(STDY3*SIG3 - CTDY3*U1/2.0)
      S(2,4)= H*(STDY3*SIG6-STD*SIG5)
      S(2,5)= H*(STDY3*SIG8-STD*SIG7)
      S(2,6)= H*STDY3*SIG9
      S(2,7)= S(2,1)
      S(2,8)= S(2,2) + 2.0
      S(2,9)= S(2,3)
      S(2,10)= S(2,4)
      S(2,11)= S(2,5)
      S(2,12)= S(2,6)

      S(3,1)= H*FY3*(P*SIG1+SIG10*U1)
      S(3,2)= H*FY3*(P*SIG2+SIG11*U1)
      S(3,3)= -1 + H*(ALPHA/2.0 + P*(U1*SIG4 + SIG3*FY3))
      S(3,4)= H*P*FY3*SIG6
      S(3,5)= H*P*FY3*SIG8
      S(3,6)= H*P*FY3*SIG9
      S(3,7)= S(3,1)
      S(3,8)= S(3,2)
      S(3,9)= S(3,3) + 2.0
      S(3,10)= S(3,4)
      S(3,11)= S(3,5)
      S(3,12)= S(3,6)

      S(4,1)= H*Y6*STD*FY3*SIG1
      S(4,2)= H*Y6*STD*FY3*SIG2
      S(4,3)= H*Y6*STD*(U1*SIG4 + SIG3*FY3)
      S(4,4)= -1 + H*Y6*STD*FY3*SIG6
      S(4,5)= H*Y6*STD*FY3*SIG8
      S(4,6)= H*STD*FY3*(Y6*SIG9 + U1/2.0)
      S(4,7)= S(4,1)
      S(4,8)= S(4,2)
      S(4,9)= S(4,3)
      S(4,10)= S(4,4) + 2.0
      S(4,11)= S(4,5)
      S(4,12)= S(4,6)

      S(5,1)= -H*Y6*CTD*FY3*SIG1
      S(5,2)= -H*Y6*CTD*FY3*SIG2
      S(5,3)= -H*Y6*CTD*(U1*SIG4 + SIG3*FY3)
      S(5,4)= -H*Y6*CTD*SIG6*FY3
      S(5,5)= -1 - H*Y6*CTD*SIG8*FY3
      S(5,6)= -H*CTD*FY3*(Y6*SIG9 + U1/2.0)
      S(5,7)= S(5,1)
      S(5,8)= S(5,2)
      S(5,9)= S(5,3)
      S(5,10)= S(5,4)
      S(5,11)= S(5,5) + 2.0
      S(5,12)= S(5,6)

      S(6,1)= -H*(Y4*STDY3*SIG1 - Y5*CTDY3*SIG1 +
     &             Y6*FPY3*(P*SIG1+SIG10*U1))
      S(6,2)= -H*(Y4*STDY3*SIG2 - Y5*CTDY3*SIG2 +
     &             Y6*FPY3*(P*SIG2+SIG11*U1))
      S(6,3)= -H*(Y4*(-U1*CTDY3/2.0 + SIG3*STDY3)
     &           - Y5*(U1*STDY3/2.0 + CTDY3*SIG3)
     &           + Y6*P*(U1*SIG12 + SIG3*FPY3))
      S(6,4)= -H*(STDY3*(Y4*SIG6 + U1/2.0) - Y5*CTDY3*SIG6 +
     &             Y6*P*FPY3*SIG6)
      S(6,5)= -H*(Y4*STDY3*SIG8 - CTDY3*(Y5*SIG8 + U1/2.0) +
     &             Y6*P*FPY3*SIG8)
      S(6,6)= -1 -H*(Y4*STDY3*SIG9 - Y5*CTDY3*SIG9
     &                 +Y6*P*FPY3*SIG9 + (ALPHA+P*U1*FPY3)/2.0)
      S(6,7)= S(6,1)
      S(6,8)= S(6,2)
      S(6,9)= S(6,3)
      S(6,10)= S(6,4)
```

```
      S(6,11)= S(6,5)
      S(6,12)= S(6,6) + 2.0

      S(1,JSF)= Y(1,K)-Y(1,K-1)-H*(CTD*UD-CTDY3*U1)
      S(2,JSF)= Y(2,K)-Y(2,K-1)-H*(STD*UD-STDY3*U1)
      S(3,JSF)= Y(3,K)-Y(3,K-1)+H*(ALPHA*Y3+P*U1*FY3)
      S(4,JSF)= Y(4,K)-Y(4,K-1)+H*Y6*U1*STD*FY3
      S(5,JSF)= Y(5,K)-Y(5,K-1)-H*Y6*U1*CTD*FY3
      S(6,JSF)= Y(6,K)-Y(6,K-1)-H*(Y4*U1*STDY3-Y5*U1*CTDY3+
     &           Y6*(ALPHA+P*U1*FPY3))

         ENDIF

C     Dummy use of variables to prevent inocculous warning on MS Compiler
C     (Variables not used)
      IS1 = IS1
      ISF = ISF
      INDEXV(1) = INDEXV(1)
      NE = NE

      RETURN
      END
```

```
      SUBROUTINE XLATOR(Y,YDOT,H,NYJ,NYK,X,THETAD,EPS,ALPHA,
     &                  X0,YO,THETAO,XF,YF,POS)
**********************************************************************
*
*       This subroutine converts the state variables delta- x,y,theta
*       into Robot variables: x,y,theta and Virtual Robot varialbles: xd, yd.
*       A simple trapezoidal integration is used to determine position and
*       energy costs by integration of the term (u1**2)/2+alpha.
*
**********************************************************************
        IMPLICIT REAL*8 (A-H, O-Z)

        PARAMETER(M=201)
        DIMENSION Y(NYJ,NYK), YDOT(5,M), X(M), POS(5,M),
     &            COST(M),NRG(M),U1TRAJ(M),UDTRAJ(M)

*       INITIAL POSITIONS & ENERGY CONDITIONS
**********************************************************************
        POS(1,1)=X0
        POS(2,1)=Y0
        POS(3,1)=THETA0
        POS(4,1)=XF
        POS(5,1)=YF

        NRG(1)=0.

        DO 10 K=1,M

          IF(ABS(Y(3,K)).GT.EPS)THEN
            FY3= SIN(Y(3,K))/Y(3,K)
          ELSE
            FY3= 1.0
          ENDIF

*       CALCULATION OF VELOCITIES U1,UD
**********************************************************************
          Pdel= Y(2,K)*COS(THETAD)-Y(1,K)*SIN(THETAD)
          U1= Y(4,K)*COS(THETAD-Y(3,K)) +
     &        Y(5,K)*SIN(THETAD-Y(3,K)) +
     &        Y(6,K)*Pdel*FY3

          U1TRAJ(K)=U1
          UD= -Y(4,K)*COS(THETAD)-Y(5,K)*SIN(THETAD)
          UDTRAJ(K)=UD

*       CALCULATION OF ENERGY COSTS
**********************************************************************
*       ENERGY COST FUCTION FOR ROBOT4 & 2
*       NRG(K)=(U1TRAJ(K)**2+UDTRAJ(K)**2)/2
*       ENERGY COST FUNCTION IN FORM FOR ROBOT5 (used for comparison)
        NRG(K)=(U1TRAJ(K)**2+UDTRAJ(K)**2)/2 + ALPHA

          YDOT(1,K)= COS(THETAD-Y(3,K))*U1
          YDOT(2,K)= SIN(THETAD-Y(3,K))*U1
          YDOT(3,K)= Pdel*U1*FY3 + ALPHA*Y(3,K)
          YDOT(4,K)= COS(THETAD)*UD
          YDOT(5,K)= SIN(THETAD)*UD
C
C       TRAPEZOIDAL INTEGRATION:
**********************************************************************
          IF(K.GT.1)THEN
            POS(1,K)=POS(1,K-1)+H*(YDOT(1,K)+YDOT(1,K-1))/2
            POS(2,K)=POS(2,K-1)+H*(YDOT(2,K)+YDOT(2,K-1))/2
            POS(3,K)=POS(3,K-1)+H*(YDOT(3,K)+YDOT(3,K-1))/2
            POS(4,K)=POS(4,K-1)+H*(YDOT(4,K)+YDOT(4,K-1))/2
            POS(5,K)=POS(5,K-1)+H*(YDOT(5,K)+YDOT(5,K-1))/2

            COST(K)=COST(K-1)+H*(NRG(K)+NRG(K-1))/2
```

```
          ENDIF

        WRITE(34,80) X(K),YDOT(1,K),YDOT(2,K),YDOT(3,K),YDOT(4,K),
     &                YDOT(5,K)
        WRITE(35,80) X(K),POS(1,K),POS(2,K),POS(3,K),POS(4,K),
     &                POS(5,K)
        WRITE(37,81) X(K),U1TRAJ(K),UDTRAJ(K),COST(K)

10      CONTINUE

80      FORMAT(2X,6F15.4)
81      FORMAT(2X,4F15.4)

      RETURN
      END
```

APPENDIX C

Program Files Specific to Robot 2

ROBOT2.FOR

DIFEQ.FOR (for Robot 2)

XLATOR2.FOR

```
      PROGRAM ROBOT2
*********************************************************************
*       Single Robot Problem with Terminal Costs
*********************************************************************
*     Source for subroutines Red, Pinvs, Solvde, and Bksub and model for
*     Difeq and Diskmain:
*          Numerical Recipes, William H. Press, et al
*********************************************************************

      IMPLICIT REAL*8 (A-H, O-Z)

      PARAMETER(NE=6,M=201,NB=3,NCI=NE,NCJ=NE-NB+1,NCK=M+1,NSI=NE,
     &    NSJ=2*NE+1, NYJ=NE, NYK=M)

      DIMENSION SCALV(NE),INDEXV(NE),Y(NE,M),C(NCI,NCJ,NCK),S(NSI,NSJ),
     &         YDOT(NE-1,M),POS(NE-1,M)
      COMMON X(M), H, DELX0, DELY0, DELTHETO, THETAD, EPS, ALPHA, CWT

*********************************************************************
*     Variable description:
*
*     GENERAL PROGRAM VARIABLES:
*--------------------------------------------------------------------
*     NE:    Number of independent equations describing system
*     M:     Number of Meshpoints, divisions of independent variable, time
*     NB:    Number of Boundary Conditions known at initial condition
*     C:     3-D Array for storage of corrections for each iteration
*                Note: largest array in program
*     NCI, NCJ, NCK:
*            dimension variables of C array, must satisfy equations
*            found in parameter statement
*     S:     array for storage of blocks of solution of Difeq.
*     NSI, NSJ:
*            dimension variables of 2-D S array, must satisfy equations
*            found in parameter statement
*     Y:     2-D array containing initial guess for each point.  This array
*            is updated by calculated corrections.  When the corrections
*            are sufficiently small, convergence is acheived.
*     X:     Array for independent variable, time. Used only for comparison
*            of dependent variables after program completes.
*     SCALV: Array of values representing the typical magnitude of the
*            dependent variables.  Used for controlling correction magnitude.
*     INDEXV:lists column ordering for variables in S array, not used in this
*            program
*     ITMAX: Maximum number of iterations
*     CONV:  Convergence criteria for corrections to Y
*     SLOWC: Controls fraction of corrections applied to Y
*         H: Increment of independent variable, divisions between mesh points
*
*     PROBLEM SPECIFIC VARIABLES:
*--------------------------------------------------------------------
*
*       X0:             Initial X coordinate of robot
*       Y0:             Initial Y coordinate of robot
*       THETA0:                 Initial angle wrt X axis of robot
*       XF:             Desired final X coordinate of robot
*       YF:             Desired final Y coordinate of robot
*       THETAD:                 Desired final angle coordinate of robot
*       DELX0:          Initial boundary condition for state variable delta-X
*       DELY0:          Initial boundary condition for state variable delta-Y
*       DELTHETO:       Initial boundary condition for state variable delta-theta
*       EPS:            Smallest value for which f(delta-x)=Sin(delta-x)/(delta-x)
*       EPS2:           DUMMY VARIABLE USED FOR CONTINUITY WITH OTHER FORMS OF
ROBOT.
*       ALPHA:          Rotational gain related to delta-theta-dot
*       ALPHAMAX:       DUMMY VARIABLE USED FOR CONTINUITY WITH OTHER FORMS OF
ROBOT.
```

```
*       CWT:            Weighting parameter for Terminal Costs
*       RL1,RL2,RL3:    Initial guess amplitude for lambda costates.
*       OHM1,OHM2,OHM3:         Initial guess frequencies for lambda costates.
*       DC1,DC2,DC3:    Initial guess d.c. offsets for lambda costates.
*       PHI1,PHI2,PHI3:         Initial guess phase lag for lambda costates.
*       SL1,SL2,SL3:    Initial guess SCALV, scale sizes, for lambda costates.
*       POS(NE-1,M):    Position Trajectory for x,y,theta, xd and yd.
*
*
************************************************************************

      OPEN(21,FILE='xplot.rob2', STATUS='UNKNOWN')
      OPEN(22,FILE='lplot.rob2', STATUS='UNKNOWN')
      OPEN(30,FILE='xiplot.rob2', STATUS='UNKNOWN')
      OPEN(31,FILE='liplot.rob2', STATUS='UNKNOWN')
      OPEN(32,FILE='itplot.rob2', STATUS='UNKNOWN')
      OPEN(33,FILE='testdat.rob2', STATUS='UNKNOWN')
      OPEN(34,FILE='dotplot.rob2', STATUS='UNKNOWN')
      OPEN(35,FILE='posplot.rob2', STATUS='UNKNOWN')
      OPEN(37,FILE='ulplot.rob2', STATUS='UNKNOWN')

      PI= 3.141592654

      PRINT *,'ENTER ITMAX, CONV, SLOWC'
      READ *, ITMAX, CONV, SLOWC
      PRINT *,'ENTER ALPHA'
      READ *, ALPHA
       print *,'DUMMY READ FOR COMPATABILITY, enter #'
       READ *, DUMMY
      PRINT *,'ENTER C, WEIGHTING PARAMTER FOR TERMINAL COST'
      READ *, CWT
      PRINT *,'ENTER EPSILON'
      READ *, EPS
       print *,'DUMMY READ FOR COMPATABILITY, enter #'
       READ *, DUMMY
      PRINT *, 'ENTER INITIAL X,Y, AND THETA(degrees)'
      READ *, X0,Y0,THETA0
      THETA0=THETA0*PI/180
      PRINT *, 'ENTER FINAL X,Y, AND THETA(degrees)'
      READ *, XF,YF,THETAD
      THETAD=THETAD*PI/180
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA AMPLITUDES'
      READ *, RL1,RL2,RL3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA FREQUENCIES'
      READ *, OHM1,OHM2,OHM3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA DC OFFSETS'
      READ *, DC1,DC2,DC3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA PHASES'
      READ *, PHI1, PHI2, PHI3
       PHI1=PHI1*PI/180
       PHI2=PHI2*PI/180
       PHI3=PHI3*PI/180
      PRINT *,'ENTER INITIAL GUESS FOR SIZE OF 3 LAMBDA VALUES'
      READ *, SL1,SL2,SL3

      DELX0=(XF-X0)
      DELY0=(YF-Y0)
      DELTHET0=(THETAD-THETA0)

      H=1./(M-1)

C     NO INDEX CHANGES NECESSARY
C     Index Scale used by SOLVDE.FOR

      INDEXV(1)=1
      INDEXV(2)=2
      INDEXV(3)=3
      INDEXV(4)=4
      INDEXV(5)=5
```

```
      INDEXV(6)=6

c     INITIAL GUESS FOR ALL POINTS, 1 - M

C     Initialize independent vector X (time)

      DO 11 K=1,M
          X(K)=(K-1)*H
 11   CONTINUE

C     Enter initial values for all meshpoints
C       NOTE: BOUNDARY CONDITONS FOR Y(1)-Y(3) ARE
C             ENTERED AT POINTS 1 AND M DURING THE INITIAL GUESS!!!
C             THESE NUMBERS MUST COINCIDE WITH ANY DESIRED B.C.!!!

C      INITIAL GUESS:
      DO 12 K=1,M
          y(1,K)=DELX0-(DELX0*(K-1)/(M-1))
          y(2,K)=DELY0-(DELY0*(K-1)/(M-1))
          y(3,K)=DELTHETO-(DELTHETO*(K-1)/(M-1))
          y(4,K)=RL1*SIN(2*PI*OHM1*(K-1)/(M-1)+PHI1)+DC1
          y(5,K)=RL2*SIN(2*PI*OHM2*(K-1)/(M-1)+PHI2)+DC2
          y(6,K)=RL3*SIN(2*PI*OHM3*(K-1)/(M-1)+PHI3)+DC3
 12   CONTINUE

C     Write initial guess to file

      DO 13 K=1,M
            WRITE(30,80) X(K),Y(1,K),Y(2,K),Y(3,K)
            WRITE(31,80) X(K),Y(4,K),Y(5,K),Y(6,K)
 13         CONTINUE

C     Scalv set to approximate size of typical values of known solution

      SCALV(1)=ABS(DELX0/2)+.01
      SCALV(2)=ABS(DELY0/2)+.01
      SCALV(3)=ABS(2*DELTHETO/M)+.01
      SCALV(4)=ABS(SL1)+.01
      SCALV(5)=ABS(SL2)+.01
      SCALV(6)=ABS(SL3)+.01

C    WRITE TEST DATA TO FILE
      WRITE(33,*) 'ITMAX =',ITMAX
      WRITE(33,*) 'CONV  =',CONV
      WRITE(33,*) 'SLOWC =',SLOWC
      WRITE(33,*) 'ALPHA =',ALPHA
      WRITE(33,*) 'CWT =',CWT
      WRITE(33,*) 'EPSILON =',EPS
      WRITE(33,*) 'INITIAL X,Y & THETA = ', X0,Y0,(THETA0*180/PI)
      WRITE(33,*) 'FINAL   X,Y & THETA = ', XF,YF,(THETAD*180/PI)
      WRITE(33,*) 'INITIAL LAMBDA VALUES', RL1,RL2,RL3
      WRITE(33,*) 'SIZE OF LAMBDA VALUES', SL1,SL2,SL3
      WRITE(33,*) 'DELX0 =', DELX0
      WRITE(33,*) 'DELY0 =', DELY0
      WRITE(33,*) 'DELTHETO =', DELTHETO

      CALL SOLVDE(ITMAX,CONV,SLOWC,SCALV,INDEXV,NE,NB,M,Y,NYJ,NYK,
     *    C,NCI,NCJ,NCK,S,NSI,NSJ)

C     Write final Y values to file:

        DO 181 k = 1,M
          WRITE(21,80) X(K),Y(1,k),Y(2,k),Y(3,k)
          WRITE(22,80) X(K),Y(4,k),Y(5,k),Y(6,k)
 181    CONTINUE

 80   FORMAT(2X,4F15.4)

      CALL XLATOR2(Y,YDOT,H,NYJ,NYK,X,THETAD,EPS,ALPHA,
```

```
&                    X0,YO,THETA0,XF,YF,POS)

   PRINT *,'PROGRAM COMPLETED'

  CLOSE(21)
  CLOSE(22)
  CLOSE(30)
  CLOSE(31)
  CLOSE(32)
  CLOSE(33)
  CLOSE(34)
  CLOSE(35)
  CLOSE(37)

  END
```

```
      SUBROUTINE DIFEQ(K,K1,K2,JSF,IS1,ISF,INDEXV,NE,S,NSI,NSJ,Y,NYJ,
     *                  NYK)
      IMPLICIT REAL*8 (A-H, O-Z)
***********************************************************************
* MODIFIED 7/18/94 TO INCLUDE TERMINAL COST AT FINA BOUNDARY CONDITION
* MODIFIED 7/21/94 FOR NON-MOVING VIRTUAL (TARGET) ROBOT
*
***********************************************************************
      PARAMETER(M=201)
      DIMENSION Y(NYJ,NYK), S(NSI,NSJ), INDEXV(NYJ)
      COMMON X(M), H, DELX0, DELY0, DELTHETO, THETAD, EPS, ALPHA, CWT


C     Initialize matrix S as 0

      DO 10 I=1,NSI
          DO 9 J=1,NSJ
              S(I,J) = 0.0
 9        CONTINUE
 10   CONTINUE
***********************************************************************
C     Initial Boundary Conditions

      IF(K.EQ.K1) THEN

C     Enter non-zero values:

          DO 11 I= 1,3
              S(3+I,6+I)=1.0
 11       CONTINUE

C    Initial values in right hand vector for initial block

          S(4,JSF)= y(1,1)-DELX0
          S(5,JSF)= y(2,1)-DELY0
          S(6,JSF)= y(3,1)-DELTHETO
***********************************************************************
C     End Boundary Conditions

      ELSE IF (K.GT.K2) THEN

C     Enter non-zero values:

         S(1,7)= CWT
         S(1,8)= 0.0
         S(1,9)= 0.0
         S(1,10)= -1.0
         S(1,11)= 0.0
         S(1,12)= 0.0

         S(2,7)= 0.0
         S(2,8)= CWT
         S(2,9)= 0.0
         S(2,10)= 0.0
         S(2,11)= -1.0
         S(2,12)= 0.0

         S(3,7)= 0.0
         S(3,8)= 0.0
         S(3,9)= CWT
         S(3,10)= 0.0
         S(3,11)= 0.0
         S(3,12)= -1.0

C    Final values in right hand vector for final block
```

```
          S(1,JSF)= Y(1,M)*CWT - Y(4,M)
          S(2,JSF)= Y(2,M)*CWT - Y(5,M)
          S(3,JSF)= Y(3,M)*CWT - Y(6,M)

***********************************************************************
C       Interior Points
C           Derived from Finite Difference Equations of Motion

        ELSE

C       Pre-calculation of commonly used variables:

        Y1=(Y(1,K)+Y(1,K-1))/2.0
        Y2=(Y(2,K)+Y(2,K-1))/2.0
        Y3=(Y(3,K)+Y(3,K-1))/2.0
        Y4=(Y(4,K)+Y(4,K-1))/2.0
        Y5=(Y(5,K)+Y(5,K-1))/2.0
        Y6=(Y(6,K)+Y(6,K-1))/2.0

          CTD=COS(THETAD)
          STD=SIN(THETAD)
          CTDY3=COS(THETAD-Y3)
          STDY3=SIN(THETAD-Y3)

           P= Y2*CTD-Y1*STD

          IF(ABS(Y3).GT.EPS)THEN
             FY3=SIN(Y3)/(Y3)
             FPY3=(COS(Y3)/Y3)-(SIN(Y3)/(Y3**2))
             SIG4=FPY3/2.0
              SIG12=(-SIN(Y3)/Y3-2.0*COS(Y3)/(Y3)**2 +
     &              2.0*SIN(Y3)/(Y3)**3)/2.0
          ELSE
             FY3=1.0
             FPY3=0.0
             SIG4=0.0
             SIG12=-1/6.0
          ENDIF

          U1=Y4*CTDY3 + Y5*STDY3 + Y6*P*FY3

          SIG1= -Y6*FY3*STD/2.0
          SIG2= Y6*FY3*CTD/2.0
          SIG3= Y4/2.0*STDY3 - Y5/2.0*CTDY3 + Y6*P*SIG4


          SIG6= CTDY3/2.0

          SIG8= STDY3/2.0
          SIG9= P*FY3/2.0
          SIG10 = -STD/2.0
          SIG11= CTD/2.0

c       Enter non-zero values:

          S(1,1)= -1 + H*CTDY3*SIG1
          S(1,2)= H*CTDY3*SIG2
           S(1,3)= H*(CTDY3*SIG3 + STDY3*U1/2.0)
          S(1,4)= H*(CTDY3*SIG6)
          S(1,5)= H*(CTDY3*SIG8)
          S(1,6)= H*CTDY3*SIG9
          S(1,7)= S(1,1) + 2.0
          S(1,8)= S(1,2)
          S(1,9)= S(1,3)
          S(1,10)= S(1,4)
          S(1,11)= S(1,5)
          S(1,12)= S(1,6)

          S(2,1)= H*STDY3*SIG1
```

```
      S(2,2)= -1 + H*STDY3*SIG2
      S(2,3)= H*(STDY3*SIG3 - CTDY3*U1/2.0)
      S(2,4)= H*(STDY3*SIG6)
      S(2,5)= H*(STDY3*SIG8)
      S(2,6)= H*STDY3*SIG9
      S(2,7)= S(2,1)
      S(2,8)= S(2,2) + 2.0
      S(2,9)= S(2,3)
      S(2,10)= S(2,4)
      S(2,11)= S(2,5)
      S(2,12)= S(2,6)

      S(3,1)= H*FY3*(P*SIG1+SIG10*U1)
      S(3,2)= H*FY3*(P*SIG2+SIG11*U1)
      S(3,3)= -1 + H*(ALPHA/2.0 + P*(U1*SIG4 + SIG3*FY3))
      S(3,4)= H*P*FY3*SIG6
      S(3,5)= H*P*FY3*SIG8
      S(3,6)= H*P*FY3*SIG9
      S(3,7)= S(3,1)
      S(3,8)= S(3,2)
       S(3,9)= S(3,3) + 2.0
       S(3,10)= S(3,4)
      S(3,11)= S(3,5)
      S(3,12)= S(3,6)

      S(4,1)= H*Y6*STD*FY3*SIG1
      S(4,2)= H*Y6*STD*FY3*SIG2
       S(4,3)= H*Y6*STD*(U1*SIG4 + SIG3*FY3)
       S(4,4)= -1 + H*Y6*STD*FY3*SIG6
      S(4,5)= H*Y6*STD*FY3*SIG8
      S(4,6)= H*STD*FY3*(Y6*SIG9 + U1/2.0)
      S(4,7)= S(4,1)
      S(4,8)= S(4,2)
      S(4,9)= S(4,3)
      S(4,10)= S(4,4) + 2.0
      S(4,11)= S(4,5)
      S(4,12)= S(4,6)

      S(5,1)= -H*Y6*CTD*FY3*SIG1
      S(5,2)= -H*Y6*CTD*FY3*SIG2
      S(5,3)= -H*Y6*CTD*(U1*SIG4 + SIG3*FY3)
      S(5,4)= -H*Y6*CTD*SIG6*FY3
      S(5,5)= -1 - H*Y6*CTD*SIG8*FY3
      S(5,6)= -H*CTD*FY3*(Y6*SIG9 + U1/2.0)
      S(5,7)= S(5,1)
      S(5,8)= S(5,2)
      S(5,9)= S(5,3)
      S(5,10)= S(5,4)
      S(5,11)= S(5,5) + 2.0
      S(5,12)= S(5,6)

      S(6,1)= -H*(Y4*STDY3*SIG1 - Y5*CTDY3*SIG1 +
&              Y6*FPY3*(P*SIG1+SIG10*U1))
      S(6,2)= -H*(Y4*STDY3*SIG2 - Y5*CTDY3*SIG2 +
&              Y6*FPY3*(P*SIG2+SIG11*U1))
      S(6,3)= -H*(Y4*(-U1*CTDY3/2.0 + SIG3*STDY3)
&              - Y5*(U1*STDY3/2.0 + CTDY3*SIG3)
&             + Y6*P*(U1*SIG12 + SIG3*FPY3))
      S(6,4)= -H*(STDY3*(Y4*SIG6 + U1/2.0) - Y5*CTDY3*SIG6 +
&             Y6*P*FPY3*SIG6)
      S(6,5)= -H*(Y4*STDY3*SIG8 - CTDY3*(Y5*SIG8 + U1/2.0) +
&             Y6*P*FPY3*SIG8)
      S(6,6)= -1 -H*(Y4*STDY3*SIG9 - Y5*CTDY3*SIG9
&                  +Y6*P*FPY3*SIG9 + (ALPHA+P*U1*FPY3)/2.0)
      S(6,7)= S(6,1)
      S(6,8)= S(6,2)
      S(6,9)= S(6,3)
      S(6,10)= S(6,4)
      S(6,11)= S(6,5)
```

```
      S(6,12)= S(6,6) + 2.0

      S(1,JSF)= Y(1,K)-Y(1,K-1)+H*(CTDY3*U1)
      S(2,JSF)= Y(2,K)-Y(2,K-1)+H*(STDY3*U1)
      S(3,JSF)= Y(3,K)-Y(3,K-1)+H*(ALPHA*Y3+P*U1*FY3)
      S(4,JSF)= Y(4,K)-Y(4,K-1)+H*Y6*U1*STD*FY3
      S(5,JSF)= Y(5,K)-Y(5,K-1)-H*Y6*U1*CTD*FY3
      S(6,JSF)= Y(6,K)-Y(6,K-1)-H*(Y4*U1*STDY3-Y5*U1*CTDY3+
&               Y6*(ALPHA+P*U1*FPY3))

          ENDIF

    Dummy use of variables to prevent inocculous warning on MS Compiler
    (Variables not used)
    IS1 = IS1
    ISF = ISF
    INDEXV(1) = INDEXV(1)
    NE = NE

    RETURN
    END
```

```
      SUBROUTINE XLATOR2(Y,YDOT,H,NYJ,NYK,X,THETAD,EPS,ALPHA,
     &                   X0,YO,THETA0,XF,YF,POS)
*****************************************************************
*****************************************************************
*      MODIFIED 7/21/94 FOR FIXED VIRTUAL (TARGET) ROBOT PROBLEM
*       SUCH THAT UD=0 FOR ALL TIME
*****************************************************************

        IMPLICIT REAL*8 (A-H, O-Z)

        PARAMETER(M=201)
        DIMENSION Y(NYJ,NYK), YDOT(5,M), X(M), POS(5,M),
     &            COST(M),NRG(M),U1TRAJ(M)

        POS(1,1)=X0
        POS(2,1)=Y0
        POS(3,1)=THETA0
        POS(4,1)=XF
        POS(5,1)=YF

        NRG(1)=0.

        GU1=1

        DO 10 K=1,M

          IF(ABS(Y(3,K)).GT.EPS)THEN
            FY3= SIN(Y(3,K))/Y(3,K)
          ELSE
            FY3= 1.0
          ENDIF

          Pdel= Y(2,K)*COS(THETAD)-Y(1,K)*SIN(THETAD)
          U1= Y(4,K)*COS(THETAD-Y(3,K)) +
     &        Y(5,K)*SIN(THETAD-Y(3,K)) +
     &        Y(6,K)*Pdel*FY3

        U1TRAJ(K)=U1
*       ENERGY COST FUCTION FOR ROBOT4 & 2
*       NRG(K)=(U1TRAJ(K)**2)/2
*       ENERGY COST FUNCTION IN FORM FOR ROBOT5 (used for comparison)
        NRG(K)=(U1TRAJ(K)**2)/2 + ALPHA

          UD= 0.0

          YDOT(1,K)= COS(THETAD-Y(3,K))*U1
          YDOT(2,K)= SIN(THETAD-Y(3,K))*U1
          YDOT(3,K)= Pdel*U1*FY3 + ALPHA*Y(3,K)
          YDOT(4,K)= COS(THETAD)*UD
          YDOT(5,K)= SIN(THETAD)*UD

          IF(K.GT.1)THEN
            POS(1,K)=XF-Y(1,K)
            POS(2,K)=YF-Y(2,K)
            POS(3,K)=THETAD-Y(3,K)
            POS(4,K)=XF
            POS(5,K)=YF
C
C       TRAPEZOIDAL INTEGRATION:
C
            COST(K)=COST(K-1)+H*(NRG(K)+NRG(K-1))/2
          ENDIF

        WRITE(34,80) X(K),YDOT(1,K),YDOT(2,K),YDOT(3,K),YDOT(4,K),
     &               YDOT(5,K)
        WRITE(35,80) X(K),POS(1,K),POS(2,K),POS(3,K),POS(4,K),
     &               POS(5,K)
        WRITE(37,81) X(K),U1TRAJ(K),GU1,COST(K)
```

```
10    CONTINUE

80    FORMAT(2X,6F15.4)
81    FORMAT(2X,4F20.10)

      RETURN
      END
```

APPENDIX D

Program Files Specific to Robot 3

ROBOT3.FOR

DIFEQ.FOR (for Robot 3)

XLATOR3.FOR

```
      PROGRAM ROBOT3
*********************************************************************
*      Single Robot with Terminal Costs and g(u1)=function of u1.
*********************************************************************
*     Source for subroutines Red, Pinvs, Solvde, and Bksub and model for
*     Difeq and Diskmain:
*          Numerical Recipes, William H. Press, et al
*********************************************************************

      IMPLICIT REAL*8 (A-H, O-Z)

      PARAMETER(NE=6,M=201,NB=3,NCI=NE,NCJ=NE-NB+1,NCK=M+1,NSI=NE,
     &    NSJ=2*NE+1, NYJ=NE, NYK=M)

      DIMENSION SCALV(NE),INDEXV(NE),Y(NE,M),C(NCI,NCJ,NCK),S(NSI,NSJ),
     &          YDOT(NE-1,M),POS(NE-1,M)
      COMMON X(M), H, DELX0, DELY0, DELTHETO, THETAD, EPS, ALPHA, CWT,
     &        EPS2

*********************************************************************
*     Variable description:
*
*     GENERAL PROGRAM VARIABLES:
*-------------------------------------------------------------------------
*     NE:     Number of independent equations describing system
*     M:      Number of Meshpoints, divisions of independent variable, time
*     NB:     Number of Boundary Conditions known at initial condition
*     C:      3-D Array for storage of corrections for each iteration
*                Note: largest array in program
*     NCI, NCJ, NCK:
*             dimension variables of C array, must satisfy equations
*             found in parameter statement
*     S:      array for storage of blocks of solution of Difeq.
*     NSI, NSJ:
*             dimension variables of 2-D S array, must satisfy equations
*             found in parameter statement
*     Y:      2-D array containing initial guess for each point.  This array
*             is updated by calculated corrections.  When the corrections
*             are sufficiently small, convergence is acheived.
*     X:      Array for independent variable, time. Used only for comparison
*             of dependent variables after program completes.
*     SCALV: Array of values representing the typical magnitude of the
*             dependent variables.  Used for controlling correction magnitude.
*     INDEXV:lists column ordering for variables in S array, not used in this
*             program
*     ITMAX: Maximum number of iterations
*     CONV:  Convergence criteria for corrections to Y
*     SLOWC: Controls fraction of corrections applied to Y
*         H: Increment of independent variable, divisions between mesh points
*
*     PROBLEM SPECIFIC VARIABLES:
*-------------------------------------------------------------------------
*
*
*       X0:              Initial X coordinate of robot
*       Y0:              Initial Y coordinate of robot
*       THETA0:                   Initial angle wrt X axis of robot
*       XF:              Desired final X coordinate of robot
*       YF:              Desired final Y coordinate of robot
*       THETAD:                   Desired final angle coordinate of robot
*       DELX0:           Initial boundary condition for state variable delta-X
*       DELY0:           Initial boundary condition for state variable delta-Y
*       DELTHETO:        Initial boundary condition for state variable delta-theta
*       EPS:             Smallest value for which f(delta-x)=Sin(delta-x)/(delta-x)
*       EPS2:            Smallest value of u1 for which g(u1)=1.
*       ALPHA:           Rotational gain related to delta-theta-dot
*       ALPHAMAX:        DUMMY VARIABLE USED FOR CONTINUITY WITH OTHER FORMS OF
```

```
ROBOT.
*       CWT:            Weighting parameter for Terminal Costs
*       RL1,RL2,RL3:    Initial guess amplitude for lambda costates.
*       OHM1,OHM2,OHM3:         Initial guess frequencies for lambda costates.
*       DC1,DC2,DC3:    Initial guess d.c. offsets for lambda costates.
*       PHI1,PHI2,PHI3:         Initial guess phase lag for lambda costates.
*       SL1,SL2,SL3:    Initial guess SCALV, scale sizes, for lambda costates.
*       POS(NE-1,M):    Position Trajectory for x,y,theta, xd and yd.
*
*
**********************************************************************

      OPEN(21,FILE='xplot.rob3', STATUS='UNKNOWN')
      OPEN(22,FILE='lplot.rob3', STATUS='UNKNOWN')
      OPEN(30,FILE='xiplot.rob3', STATUS='UNKNOWN')
      OPEN(31,FILE='liplot.rob3', STATUS='UNKNOWN')
      OPEN(32,FILE='itplot.rob3', STATUS='UNKNOWN')
      OPEN(33,FILE='testdat.rob3', STATUS='UNKNOWN')
      OPEN(34,FILE='dotplot.rob3', STATUS='UNKNOWN')
      OPEN(35,FILE='posplot.rob3', STATUS='UNKNOWN')
      OPEN(37,FILE='ulplot.rob3', STATUS='UNKNOWN')

      PI= 3.141592654

      PRINT *,'ENTER ITMAX, CONV, SLOWC'
      READ *, ITMAX, CONV, SLOWC
      PRINT *,'ENTER ALPHA'
      READ *, ALPHA
*       Dummy read for compatability with standard data input
        print *, 'enter dummy number'
        READ *, dummy
      PRINT *,'ENTER C, WEIGHTING PARAMTER FOR TERMINAL COST'
      READ *, CWT
      PRINT *,'ENTER EPSILON'
      READ *, EPS
      PRINT *,'ENTER EPSILON 2'
      READ *, EPS2
      PRINT *, 'ENTER INITIAL X,Y, AND THETA(degrees)'
      READ *, X0,Y0,THETA0
      THETA0=THETA0*PI/180
      PRINT *, 'ENTER FINAL X,Y, AND THETA(degrees)'
      READ *, XF,YF,THETAD
      THETAD=THETAD*PI/180
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA AMPLITUDES'
      READ *, RL1,RL2,RL3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA FREQUENCIES'
      READ *, OHM1,OHM2,OHM3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA DC OFFSETS'
      READ *, DC1,DC2,DC3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA PHASES'
      READ *, PHI1, PHI2, PHI3
        PHI1=PHI1*PI/180
        PHI2=PHI2*PI/180
        PHI3=PHI3*PI/180
      PRINT *,'ENTER INITIAL GUESS FOR SIZE OF 3 LAMBDA VALUES'
      READ *, SL1,SL2,SL3

      DELX0=(XF-X0)
      DELY0=(YF-Y0)
      DELTHET0=(THETAD-THETA0)

      H=1./(M-1)

C     NO INDEX CHANGES NECESSARY
C     Index Scale used by SOLVDE.FOR

      INDEXV(1)=1
      INDEXV(2)=2
      INDEXV(3)=3
```

```
      INDEXV(4)=4
      INDEXV(5)=5
      INDEXV(6)=6

c     INITIAL GUESS FOR ALL POINTS, 1 - M

C     Initialize independent vector X (time)

      DO 11 K=1,M
         X(K)=(K-1)*H
 11   CONTINUE

C      INITIAL GUESS:

C     Enter initial values for all meshpoints
C       NOTE: BOUNDARY CONDITONS FOR Y(1)-Y(3) ARE
C             ENTERED AT POINTS 1 AND M DURING THE INITIAL GUESS!!!
C             THESE NUMBERS MUST COINCIDE WITH ANY DESIRED B.C.!!!

       WRITE(33,*) 'STRAIGHT LINE GUESS'
        DO 12 K=1,M
          y(1,K)=DELX0-(DELX0*(K-1)/(M-1))
          y(2,K)=DELY0-(DELY0*(K-1)/(M-1))
          y(3,K)=DELTHETO-(DELTHETO*(K-1)/(M-1))
          y(4,K)=RL1*SIN(2*PI*OHM1*(K-1)/(M-1)+PHI1)+DC1
          y(5,K)=RL2*SIN(2*PI*OHM2*(K-1)/(M-1)+PHI2)+DC2
          y(6,K)=RL3*SIN(2*PI*OHM3*(K-1)/(M-1)+PHI3)+DC3
 12     CONTINUE

C     Write initial guess to file

      DO 13 K=1,M
           WRITE(30,80) X(K),Y(1,K),Y(2,K),Y(3,K)
           WRITE(31,80) X(K),Y(4,K),Y(5,K),Y(6,K)
 13        CONTINUE

C     Scalv set to approximate size of typical values of known solution

      SCALV(1)=ABS(DELX0/2)+.01
      SCALV(2)=ABS(DELY0/2)+.01
      SCALV(3)=ABS(2*DELTHETO/M)+.01
      SCALV(4)=ABS(SL1)+.01
      SCALV(5)=ABS(SL2)+.01
      SCALV(6)=ABS(SL3)+.01

C    WRITE TEST DATA TO FILE
      WRITE(33,*) 'ITMAX =',ITMAX
      WRITE(33,*) 'CONV  =',CONV
      WRITE(33,*) 'SLOWC =',SLOWC
      WRITE(33,*) 'ALPHA =',ALPHA
      WRITE(33,*) 'CWT =',CWT
      WRITE(33,*) 'EPSILON =',EPS
      WRITE(33,*) 'EPSILON 2 =',EPS2
      WRITE(33,*) 'INITIAL X,Y & THETA = ', X0,Y0,(THETA0*180/PI)
      WRITE(33,*) 'FINAL   X,Y & THETA = ', XF,YF,(THETAD*180/PI)
      WRITE(33,*) 'INITIAL LAMBDA VALUES', RL1,RL2,RL3
      WRITE(33,*) 'SIZE OF LAMBDA VALUES', SL1,SL2,SL3
      WRITE(33,*) 'DELX0 =', DELX0
      WRITE(33,*) 'DELY0 =', DELY0
      WRITE(33,*) 'DELTHETO =', DELTHETO

      CALL SOLVDE(ITMAX,CONV,SLOWC,SCALV,INDEXV,NE,NB,M,Y,NYJ,NYK,
     *     C,NCI,NCJ,NCK,S,NSI,NSJ)

C     Write final Y values to file:

        DO 181 k = 1,M
          WRITE(21,80) X(K),Y(1,k),Y(2,k),Y(3,k)
          WRITE(22,80) X(K),Y(4,k),Y(5,k),Y(6,k)
```

```
181    CONTINUE

80   FORMAT(2X,4F15.4)

     CALL XLATOR3(Y,YDOT,H,NYJ,NYK,X,THETAD,EPS,ALPHA,
    &                 X0,Y0,THETA0,XF,YF,POS,EPS2)

      PRINT *,'PROGRAM COMPLETED'

     CLOSE(21)
     CLOSE(22)
     CLOSE(30)
     CLOSE(31)
     CLOSE(32)
     CLOSE(33)
     CLOSE(34)
     CLOSE(35)
     CLOSE(37)

     END
```

```
      SUBROUTINE DIFEQ(K,K1,K2,JSF,IS1,ISF,INDEXV,NE,S,NSI,NSJ,Y,NYJ,
     *                 NYK)
      IMPLICIT REAL*8 (A-H, O-Z)

*********************************************************************
* MODIFIED 7/18/94 TO INCLUDE TERMINAL COST AT FINA BOUNDARY CONDITION
* MODIFIED 7/21/94 FOR NON-MOVING VIRTUAL (TARGET) ROBOT
* MODIFIED 7/25/94 FOR FUNCTION, G(U1) TIMES ALPHA TERM IN U2.
*                        (FEEDBACK REFINEMENT)
*********************************************************************
      PARAMETER(M=201)
      DIMENSION Y(NYJ,NYK), S(NSI,NSJ), INDEXV(NYJ)
      COMMON X(M), H, DELX0, DELY0, DELTHETO, THETAD, EPS, ALPHA, CWT,
     &       EPS2


C     Initialize matrix S as 0

      DO 10 I=1,NSI
          DO 9 J=1,NSJ
             S(I,J) = 0.0
 9        CONTINUE
 10   CONTINUE

*********************************************************************
C     Initial Boundary Conditions

      IF(K.EQ.K1) THEN

C     Enter non-zero values:

          DO 11 I= 1,3
             S(3+I,6+I)=1.0
 11       CONTINUE

C    Initial values in right hand vector for initial block

          S(4,JSF)= y(1,1)-DELX0
          S(5,JSF)= y(2,1)-DELY0
          S(6,JSF)= y(3,1)-DELTHETO

*********************************************************************
C     End Boundary Conditions

      ELSE IF (K.GT.K2) THEN

C     Enter non-zero values:

          S(1,7)= CWT
          S(1,8)= 0.0
          S(1,9)= 0.0
          S(1,10)= -1.0
          S(1,11)= 0.0
          S(1,12)= 0.0

          S(2,7)= 0.0
          S(2,8)= CWT
          S(2,9)= 0.0
          S(2,10)= 0.0
          S(2,11)= -1.0
          S(2,12)= 0.0

          S(3,7)= 0.0
          S(3,8)= 0.0
          S(3,9)= CWT
          S(3,10)= 0.0
          S(3,11)= 0.0
          S(3,12)= -1.0
```

```
C    Final values in right hand vector for final block

          S(1,JSF)= Y(1,M)*CWT - Y(4,M)
          S(2,JSF)= Y(2,M)*CWT - Y(5,M)
          S(3,JSF)= Y(3,M)*CWT - Y(6,M)

*********************************************************************
C      Interior Points
C          Derived from Finite Difference Equations of Motion

      ELSE

C      Pre-calculation of commonly used variables:

      Y1=(Y(1,K)+Y(1,K-1))/2.0
      Y2=(Y(2,K)+Y(2,K-1))/2.0
      Y3=(Y(3,K)+Y(3,K-1))/2.0
      Y4=(Y(4,K)+Y(4,K-1))/2.0
      Y5=(Y(5,K)+Y(5,K-1))/2.0
      Y6=(Y(6,K)+Y(6,K-1))/2.0

        CTD=COS(THETAD)
        STD=SIN(THETAD)
        CTDY3=COS(THETAD-Y3)
        STDY3=SIN(THETAD-Y3)

         P= Y2*CTD-Y1*STD

        IF(ABS(Y3).GT.EPS)THEN
           FY3=SIN(Y3)/(Y3)
           FPY3=(COS(Y3)/Y3)-(SIN(Y3)/(Y3**2))
           SIG4=FPY3/2.0
            SIG12=(-SIN(Y3)/Y3-2.0*COS(Y3)/(Y3)**2 +
     &             2.0*SIN(Y3)/(Y3)**3)/2.0
        ELSE
           FY3=1.0
           FPY3=0.0
           SIG4=0.0
           SIG12=-1/6.0
        ENDIF

        U1=Y4*CTDY3 + Y5*STDY3 + Y6*P*FY3

        IF(ABS(U1).GT.EPS2)THEN
           GU1=1.0
        ELSE
           GU1=0.0
        ENDIF

        SIG1= -Y6*FY3*STD/2.0
        SIG2= Y6*FY3*CTD/2.0
        SIG3= Y4/2.0*STDY3 - Y5/2.0*CTDY3 + Y6*P*SIG4


        SIG6= CTDY3/2.0

        SIG8= STDY3/2.0
        SIG9= P*FY3/2.0
        SIG10 = -STD/2.0
        SIG11= CTD/2.0

c      Enter non-zero values:

        S(1,1)= -1 + H*CTDY3*SIG1
        S(1,2)= H*CTDY3*SIG2
         S(1,3)= H*(CTDY3*SIG3 + STDY3*U1/2.0)
        S(1,4)= H*(CTDY3*SIG6)
        S(1,5)= H*(CTDY3*SIG8)
        S(1,6)= H*CTDY3*SIG9
```

```
      S(1,7)= S(1,1) + 2.0
      S(1,8)= S(1,2)
      S(1,9)= S(1,3)
      S(1,10)= S(1,4)
      S(1,11)= S(1,5)
      S(1,12)= S(1,6)

      S(2,1)= H*STDY3*SIG1
      S(2,2)= -1 + H*STDY3*SIG2
      S(2,3)= H*(STDY3*SIG3 - CTDY3*U1/2.0)
      S(2,4)= H*(STDY3*SIG6)
      S(2,5)= H*(STDY3*SIG8)
      S(2,6)= H*STDY3*SIG9
      S(2,7)= S(2,1)
      S(2,8)= S(2,2) + 2.0
      S(2,9)= S(2,3)
      S(2,10)= S(2,4)
      S(2,11)= S(2,5)
      S(2,12)= S(2,6)

      S(3,1)= H*FY3*(P*SIG1+SIG10*U1)
      S(3,2)= H*FY3*(P*SIG2+SIG11*U1)
      S(3,3)= -1 + H*(ALPHA*GU1/2.0 + P*(U1*SIG4 + SIG3*FY3))
      S(3,4)= H*P*FY3*SIG6
      S(3,5)= H*P*FY3*SIG8
      S(3,6)= H*P*FY3*SIG9
      S(3,7)= S(3,1)
      S(3,8)= S(3,2)
      S(3,9)= S(3,3) + 2.0
      S(3,10)= S(3,4)
      S(3,11)= S(3,5)
      S(3,12)= S(3,6)

      S(4,1)= H*Y6*STD*FY3*SIG1
      S(4,2)= H*Y6*STD*FY3*SIG2
      S(4,3)= H*Y6*STD*(U1*SIG4 + SIG3*FY3)
      S(4,4)= -1 + H*Y6*STD*FY3*SIG6
      S(4,5)= H*Y6*STD*FY3*SIG8
      S(4,6)= H*STD*FY3*(Y6*SIG9 + U1/2.0)
      S(4,7)= S(4,1)
      S(4,8)= S(4,2)
      S(4,9)= S(4,3)
      S(4,10)= S(4,4) + 2.0
      S(4,11)= S(4,5)
      S(4,12)= S(4,6)

      S(5,1)= -H*Y6*CTD*FY3*SIG1
      S(5,2)= -H*Y6*CTD*FY3*SIG2
      S(5,3)= -H*Y6*CTD*(U1*SIG4 + SIG3*FY3)
      S(5,4)= -H*Y6*CTD*SIG6*FY3
      S(5,5)= -1 - H*Y6*CTD*SIG8*FY3
      S(5,6)= -H*CTD*FY3*(Y6*SIG9 + U1/2.0)
      S(5,7)= S(5,1)
      S(5,8)= S(5,2)
      S(5,9)= S(5,3)
      S(5,10)= S(5,4)
      S(5,11)= S(5,5) + 2.0
      S(5,12)= S(5,6)

      S(6,1)= -H*(Y4*STDY3*SIG1 - Y5*CTDY3*SIG1 +
     &            Y6*FPY3*(P*SIG1+SIG10*U1))
      S(6,2)= -H*(Y4*STDY3*SIG2 - Y5*CTDY3*SIG2 +
     &            Y6*FPY3*(P*SIG2+SIG11*U1))
      S(6,3)= -H*(Y4*(-U1*CTDY3/2.0 + SIG3*STDY3)
     &           - Y5*(U1*STDY3/2.0 + CTDY3*SIG3)
     &           + Y6*P*(U1*SIG12 + SIG3*FPY3))
      S(6,4)= -H*(STDY3*(Y4*SIG6 + U1/2.0) - Y5*CTDY3*SIG6 +
     &            Y6*P*FPY3*SIG6)
      S(6,5)= -H*(Y4*STDY3*SIG8 - CTDY3*(Y5*SIG8 + U1/2.0) +
```

```
     &               Y6*P*FPY3*SIG8)
        S(6,6)= -1 -H*(Y4*STDY3*SIG9 - Y5*CTDY3*SIG9
     &                  +Y6*P*FPY3*SIG9 + (ALPHA*GU1+P*U1*FPY3)/2.0)
        S(6,7)= S(6,1)
        S(6,8)= S(6,2)
        S(6,9)= S(6,3)
        S(6,10)= S(6,4)
        S(6,11)= S(6,5)
        S(6,12)= S(6,6) + 2.0

        S(1,JSF)= Y(1,K)-Y(1,K-1)+H*(CTDY3*U1)
        S(2,JSF)= Y(2,K)-Y(2,K-1)+H*(STDY3*U1)
        S(3,JSF)= Y(3,K)-Y(3,K-1)+H*(ALPHA*GU1*Y3+P*U1*FY3)
        S(4,JSF)= Y(4,K)-Y(4,K-1)+H*Y6*U1*STD*FY3
        S(5,JSF)= Y(5,K)-Y(5,K-1)-H*Y6*U1*CTD*FY3
        S(6,JSF)= Y(6,K)-Y(6,K-1)-H*(Y4*U1*STDY3-Y5*U1*CTDY3+
     &             Y6*(ALPHA*GU1+P*U1*FPY3))

          ENDIF

C     Dummy use of variables to prevent inocculous warning on MS Compiler
C     (Variables not used)
      IS1 = IS1
      ISF = ISF
      INDEXV(1) = INDEXV(1)
      NE = NE

      RETURN
      END
```

```
      SUBROUTINE XLATOR3(Y,YDOT,H,NYJ,NYK,X,THETAD,EPS,ALPHA,
     &                   X0,YO,THETA0,XF,YF,POS,EPS2)
******************************************************************
******************************************************************
*     MODIFIED 7/21/94 FOR FIXED VIRTUAL (TARGET) ROBOT PROBLEM
*      SUCH THAT UD=0 FOR ALL TIME

* MODIFIED 7/25/94 FOR FUNCTION, G(U1) TIMES ALPHA TERM IN U2.
*                        (FEEDBACK REFINEMENT)
******************************************************************

      IMPLICIT REAL*8 (A-H, O-Z)

      PARAMETER(M=201)
      DIMENSION Y(NYJ,NYK), YDOT(5,M), X(M), POS(5,M),
     &          COST(M),NRG(M),U1TRAJ(M),GU1TRAJ(M)

      POS(1,1)=X0
      POS(2,1)=Y0
      POS(3,1)=THETA0
      POS(4,1)=XF
      POS(5,1)=YF

      NRG(1)=0.

      DO 10 K=1,M

        IF(ABS(Y(3,K)).GT.EPS)THEN
          FY3= SIN(Y(3,K))/Y(3,K)
        ELSE
          FY3= 1.0
        ENDIF

        Pdel= Y(2,K)*COS(THETAD)-Y(1,K)*SIN(THETAD)
        U1= Y(4,K)*COS(THETAD-Y(3,K)) +
     &      Y(5,K)*SIN(THETAD-Y(3,K)) +
     &      Y(6,K)*Pdel*FY3

      U1TRAJ(K)=U1
*     ENERGY COST FUCTION FOR ROBOT4 & 2
*     NRG(K)=(U1TRAJ(K)**2)/2
*     ENERGY COST FUNCTION IN FORM FOR ROBOT5 (used for comparison)
      NRG(K)=(U1TRAJ(K)**2)/2 + ALPHA


      IF(ABS(U1).GT.EPS2)THEN
        GU1=1.0
      ELSE
        GU1=0.0
      ENDIF

      GU1TRAJ(K)=GU1

        UD= 0.0

        YDOT(1,K)= COS(THETAD-Y(3,K))*U1
        YDOT(2,K)= SIN(THETAD-Y(3,K))*U1
        YDOT(3,K)= Pdel*U1*FY3 + ALPHA*GU1*Y(3,K)
        YDOT(4,K)= COS(THETAD)*UD
        YDOT(5,K)= SIN(THETAD)*UD

        IF(K.GT.1)THEN
          POS(1,K)=XF-Y(1,K)
          POS(2,K)=YF-Y(2,K)
          POS(3,K)=THETAD-Y(3,K)
          POS(4,K)=XF
          POS(5,K)=YF
C
```

```
C       TRAPEZOIDAL INTEGRATION:
C
             COST(K)=COST(K-1)+H*(NRG(K)+NRG(K-1))/2
          ENDIF


        WRITE(34,80) X(K),YDOT(1,K),YDOT(2,K),YDOT(3,K),YDOT(4,K),
     &              YDOT(5,K)
        WRITE(35,80) X(K),POS(1,K),POS(2,K),POS(3,K),POS(4,K),
     &              POS(5,K)
        WRITE(37,81) X(K),U1TRAJ(K),GU1TRAJ(K),COST(K)

 10     CONTINUE

 80     FORMAT(2X,6F20.10)
 81     FORMAT(2X,4F20.10)

      RETURN
      END
```

APPENDIX E

Program Files Specific to Robot 4

ROBOT4.FOR

DIFEQ.FOR (for Robot 4)

XLATOR4.FOR

```
      PROGRAM ROBOT4
*******************************************************************
*      Single Robot with terminal cost and g(U1) and alpha control (hi/lo)
*******************************************************************
*     Source for subroutines Red, Pinvs, Solvde, and Bksub and model for
*     Difeq and Diskmain:
*          Numerical Recipes, William H. Press, et al
*******************************************************************

      IMPLICIT REAL*8 (A-H, O-Z)

      PARAMETER(NE=6,M=201,NB=3,NCI=NE,NCJ=NE-NB+1,NCK=M+1,NSI=NE,
     &     NSJ=2*NE+1, NYJ=NE, NYK=M)

      DIMENSION SCALV(NE),INDEXV(NE),Y(NE,M),C(NCI,NCJ,NCK),S(NSI,NSJ),
     &          YDOT(NE-1,M),POS(NE-1,M)
      COMMON X(M), H, DELX0, DELY0, DELTHETO, THETAD, EPS, CWT,ALPHA(M),
     &       EPS2, ALPHAMIN, ALPHAMAX

*******************************************************************
*     Variable description:
*
*     GENERAL PROGRAM VARIABLES:
*-------------------------------------------------------------------
*     NE:    Number of independent equations describing system
*     M:     Number of Meshpoints, divisions of independent variable, time
*     NB:    Number of Boundary Conditions known at initial condition
*     C:     3-D Array for storage of corrections for each iteration
*                 Note: largest array in program
*     NCI, NCJ, NCK:
*            dimension variables of C array, must satisfy equations
*            found in parameter statement
*     S:     array for storage of blocks of solution of Difeq.
*     NSI, NSJ:
*            dimension variables of 2-D S array, must satisfy equations
*            found in parameter statement
*     Y:     2-D array containing initial guess for each point.  This array
*            is updated by calculated corrections.  When the corrections
*            are sufficiently small, convergence is acheived.
*     X:     Array for independent variable, time. Used only for comparison
*            of dependent variables after program completes.
*     SCALV: Array of values representing the typical magnitude of the
*            dependent variables.  Used for controlling correction magnitude.
*     INDEXV:lists column ordering for variables in S array, not used in this
*            program
*     ITMAX: Maximum number of iterations
*     CONV:  Convergence criteria for corrections to Y
*     SLOWC: Controls fraction of corrections applied to Y
*         H: Increment of independent variable, divisions between mesh points
*
*     PROBLEM SPECIFIC VARIABLES:
*-------------------------------------------------------------------
*
*       X0:             Initial X coordinate of robot
*       Y0:             Initial Y coordinate of robot
*       THETA0:                 Initial angle wrt X axis of robot
*       XF:             Desired final X coordinate of robot
*       YF:             Desired final Y coordinate of robot
*       THETAD:                 Desired final angle coordinate of robot
*       DELX0:          Initial boundary condition for state variable delta-X
*       DELY0:          Initial boundary condition for state variable delta-Y
*       DELTHETO:       Initial boundary condition for state variable delta-theta
*       EPS:            Smallest value for which f(delta-x)=Sin(delta-x)/(delta-x)
*       EPS2:           Smallest value U1 for which g(u1) = 1
*       ALPHAMIN:       Low alpha gain for u1 equal to 0
*       ALPHAMAX:       High alpha gain for u1 not equal to 0
*       CWT:            Weighting parameter for Terminal Costs
*       RL1,RL2,RL3:    Initial guess amplitude for lambda costates.
*       OHM1,OHM2,OHM3:         Initial guess frequencies for lambda costates.
```

```
*       DC1,DC2,DC3:   Initial guess d.c. offsets for lambda costates.
*       PHI1,PHI2,PHI3:        Initial guess phase lag for lambda costates.
*       SL1,SL2,SL3:   Initial guess SCALV, scale sizes, for lambda costates.
*       POS(NE-1,M):   Position Trajectory for x,y,theta, xd and yd.
*
*
*********************************************************************

      OPEN(21,FILE='xplot.rob4', STATUS='UNKNOWN')
      OPEN(22,FILE='lplot.rob4', STATUS='UNKNOWN')
      OPEN(30,FILE='xiplot.rob4', STATUS='UNKNOWN')
      OPEN(31,FILE='liplot.rob4', STATUS='UNKNOWN')
      OPEN(32,FILE='itplot.rob4', STATUS='UNKNOWN')
      OPEN(33,FILE='testdat.rob4', STATUS='UNKNOWN')
      OPEN(34,FILE='dotplot.rob4', STATUS='UNKNOWN')
      OPEN(35,FILE='posplot.rob4', STATUS='UNKNOWN')
      OPEN(36,FILE='alplot.rob4', STATUS='UNKNOWN')
      OPEN(37,FILE='ulplot.rob4', STATUS='UNKNOWN')

      PI= 3.141592654

      PRINT *,'ENTER ITMAX, CONV, SLOWC'
      READ *, ITMAX, CONV, SLOWC
      PRINT *,'ENTER ALPHAMIN'
      READ *, ALPHA MIN
      PRINT *,'ENTER ALPHAMAX'
      READ *, ALPHA MAX
      PRINT *,'ENTER C, WEIGHTING PARAMTER FOR TERMINAL COST'
      READ *, CWT
      PRINT *,'ENTER EPSILON'
      READ *, EPS
      PRINT *,'ENTER EPSILON 2'
      READ *, EPS2
      PRINT *, 'ENTER INITIAL X,Y, AND THETA(degrees)'
      READ *, X0,Y0,THETA0
      THETA0=THETA0*PI/180
      PRINT *, 'ENTER FINAL X,Y, AND THETA(degrees)'
      READ *, XF,YF,THETAD
      THETAD=THETAD*PI/180
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA AMPLITUDES'
      READ *, RL1,RL2,RL3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA FREQUENCIES'
      READ *, OHM1,OHM2,OHM3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA DC OFFSETS'
      READ *, DC1,DC2,DC3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA PHASES'
      READ *, PHI1, PHI2, PHI3
       PHI1=PHI1*PI/180
       PHI2=PHI2*PI/180
       PHI3=PHI3*PI/180
      PRINT *,'ENTER INITIAL GUESS FOR SIZE OF 3 LAMBDA VALUES'
      READ *, SL1,SL2,SL3

      DELX0=(XF-X0)
      DELY0=(YF-Y0)
      DELTHET0=(THETAD-THETA0)

      H=1./(M-1)

C     NO INDEX CHANGES NECESSARY
C     Index Scale used by SOLVDE.FOR

      INDEXV(1)=1
      INDEXV(2)=2
      INDEXV(3)=3
      INDEXV(4)=4
      INDEXV(5)=5
      INDEXV(6)=6
```

```
c     INITIAL GUESS FOR ALL POINTS, 1 - M

C     Initialize independent vector X (time)

      DO 11 K=1,M
          X(K)=(K-1)*H
 11   CONTINUE

C      INITIAL GUESS:

C     Enter initial values for all meshpoints
C       NOTE: BOUNDARY CONDITONS FOR Y(1)-Y(3) ARE
C             ENTERED AT POINTS 1 AND M DURING THE INITIAL GUESS!!!
C             THESE NUMBERS MUST COINCIDE WITH ANY DESIRED B.C.!!!

       WRITE(33,*) 'STRAIGHT LINE GUESS'
       DO 12 K=1,M
         y(1,K)=DELXO-(DELXO*(K-1)/(M-1))
         y(2,K)=DELYO-(DELYO*(K-1)/(M-1))
         y(3,K)=DELTHETO-(DELTHETO*(K-1)/(M-1))
         y(4,K)=RL1*SIN(2*PI*OHM1*(K-1)/(M-1)+PHI1)+DC1
         y(5,K)=RL2*SIN(2*PI*OHM2*(K-1)/(M-1)+PHI2)+DC2
         y(6,K)=RL3*SIN(2*PI*OHM3*(K-1)/(M-1)+PHI3)+DC3
 12     CONTINUE

*       INITIAL ALPHA DETERMINATION:
*********************************************************************
       IF(Y(3,1).GT.EPS) THEN
          FY3= SIN(Y(3,1))/(Y(3,1))
       ELSE
          FY3= 1.0
       ENDIF

       U10=Y(4,1)*COS(THETAD-Y(3,1)) + Y(5,1)*SIN(THETAD-Y(3,1))
     &    + Y(6,1)*(Y(2,1)*COS(THETAD)-Y(1,1)*SIN(THETAD))*FY3

       IF(ABS(U10).GT.EPS2)THEN
          GU1=1.0
       ELSE
          GU1=0.0
       ENDIF

       BETA=1-Y(6,1)*GU1*Y(3,1)

       IF(BETA.GT.(0.0))THEN
          ALPHA(1)=ALPHAMIN
       ELSE
          ALPHA(1)=ALPHAMAX
       ENDIF
*********************************************************************

C     Write initial guess to file

      DO 13 K=1,M
           WRITE(30,80) X(K),Y(1,K),Y(2,K),Y(3,K)
           WRITE(31,80) X(K),Y(4,K),Y(5,K),Y(6,K)
 13        CONTINUE

C     Scalv set to approximate size of typical values of known solution

      SCALV(1)=ABS(DELXO/2)+.01
      SCALV(2)=ABS(DELYO/2)+.01
      SCALV(3)=ABS(2*DELTHETO/M)+.01
      SCALV(4)=ABS(SL1)+.01
      SCALV(5)=ABS(SL2)+.01
      SCALV(6)=ABS(SL3)+.01

C    WRITE TEST DATA TO FILE
     WRITE(33,*) 'ITMAX =',ITMAX
```

```
      WRITE(33,*) 'CONV  =',CONV
      WRITE(33,*) 'SLOWC =',SLOWC
      WRITE(33,*) 'ALPHA MIN =',ALPHAMIN
      WRITE(33,*) 'ALPHA MAX =',ALPHAMAX
      WRITE(33,*) 'CWT =',CWT
      WRITE(33,*) 'EPSILON =',EPS
      WRITE(33,*) 'EPSILON 2 =',EPS2
      WRITE(33,*) 'INITIAL X,Y & THETA = ', X0,Y0,(THETA0*180/PI)
      WRITE(33,*) 'FINAL   X,Y & THETA = ', XF,YF,(THETAD*180/PI)
      WRITE(33,*) 'INITIAL LAMBDA VALUE AMPLITUDE', RL1,RL2,RL3
      WRITE(33,*) 'SIZE OF LAMBDA VALUES', SL1,SL2,SL3
      WRITE(33,*) 'DELX0 =', DELX0
      WRITE(33,*) 'DELY0 =', DELY0
      WRITE(33,*) 'DELTHETO =', DELTHETO

      CALL SOLVDE(ITMAX,CONV,SLOWC,SCALV,INDEXV,NE,NB,M,Y,NYJ,NYK,
     *     C,NCI,NCJ,NCK,S,NSI,NSJ)

C     Write final Y values to file:

        DO 181 k = 1,M
          WRITE(21,80) X(K),Y(1,k),Y(2,k),Y(3,k)
          WRITE(22,80) X(K),Y(4,k),Y(5,k),Y(6,k)
         WRITE(36,81) X(K),ALPHA(K)
 181    CONTINUE

 80   FORMAT(2X,4F15.4)
 81   FORMAT(2X,2F15.4)

      CALL XLATOR4(Y,YDOT,H,NYJ,NYK,X,THETAD,EPS,ALPHA,
     &                  X0,Y0,THETA0,XF,YF,POS,EPS2)

       PRINT *,'PROGRAM COMPLETED'

      CLOSE(21)
      CLOSE(22)
      CLOSE(30)
      CLOSE(31)
      CLOSE(32)
      CLOSE(33)
      CLOSE(34)
      CLOSE(35)
      CLOSE(36)
      CLOSE(37)

      END
```

```
      SUBROUTINE DIFEQ(K,K1,K2,JSF,IS1,ISF,INDEXV,NE,S,NSI,NSJ,Y,NYJ,
     *                 NYK)
      IMPLICIT REAL*8 (A-H, O-Z)

*******************************************************************
* MODIFIED 7/18/94 TO INCLUDE TERMINAL COST AT FINA BOUNDARY CONDITION
* MODIFIED 7/21/94 FOR NON-MOVING VIRTUAL (TARGET) ROBOT
* MODIFIED 7/25/94 FOR FUNCTION, G(U1) TIMES ALPHA TERM IN U2.
*                   (FEEDBACK REFINEMENT)
* MODIFIED 7/26/94 FOR ALPHA AS INPUT
*******************************************************************
      PARAMETER(M=201)
      DIMENSION Y(NYJ,NYK), S(NSI,NSJ), INDEXV(NYJ)
      COMMON X(M), H, DELX0, DELY0, DELTHETO, THETAD, EPS, CWT,ALPHA(M),
     &       EPS2, ALPHAMIN, ALPHAMAX


C     Initialize matrix S as 0

      DO 10 I=1,NSI
          DO 9 J=1,NSJ
             S(I,J) = 0.0
 9        CONTINUE
 10   CONTINUE

*******************************************************************
C     Initial Boundary Conditions

      IF(K.EQ.K1) THEN

C     Enter non-zero values:

          DO 11 I= 1,3
             S(3+I,6+I)=1.0
 11       CONTINUE

C    Initial values in right hand vector for initial block

          S(4,JSF)= y(1,1)-DELX0
          S(5,JSF)= y(2,1)-DELY0
          S(6,JSF)= y(3,1)-DELTHETO

*******************************************************************
C     End Boundary Conditions

      ELSE IF (K.GT.K2) THEN

C     Enter non-zero values:

          S(1,7)= CWT
          S(1,8)= 0.0
          S(1,9)= 0.0
          S(1,10)= -1.0
          S(1,11)= 0.0
          S(1,12)= 0.0

          S(2,7)= 0.0
          S(2,8)= CWT
          S(2,9)= 0.0
          S(2,10)= 0.0
          S(2,11)= -1.0
          S(2,12)= 0.0

          S(3,7)= 0.0
          S(3,8)= 0.0
          S(3,9)= CWT
          S(3,10)= 0.0
          S(3,11)= 0.0
          S(3,12)= -1.0
```

```
C    Final values in right hand vector for final block

          S(1,JSF)= Y(1,M)*CWT - Y(4,M)
          S(2,JSF)= Y(2,M)*CWT - Y(5,M)
          S(3,JSF)= Y(3,M)*CWT - Y(6,M)

************************************************************************
C      Interior Points
C          Derived from Finite Difference Equations of Motion

       ELSE

C      Pre-calculation of commonly used variables:

       Y1=(Y(1,K)+Y(1,K-1))/2.0
       Y2=(Y(2,K)+Y(2,K-1))/2.0
       Y3=(Y(3,K)+Y(3,K-1))/2.0
       Y4=(Y(4,K)+Y(4,K-1))/2.0
       Y5=(Y(5,K)+Y(5,K-1))/2.0
       Y6=(Y(6,K)+Y(6,K-1))/2.0

         CTD=COS(THETAD)
         STD=SIN(THETAD)
         CTDY3=COS(THETAD-Y3)
         STDY3=SIN(THETAD-Y3)

          P= Y2*CTD-Y1*STD

         IF(ABS(Y3).GT.EPS)THEN
            FY3=SIN(Y3)/(Y3)
            FPY3=(COS(Y3)/Y3)-(SIN(Y3)/(Y3**2))
            SIG4=FPY3/2.0
             SIG12=(-SIN(Y3)/Y3-2.0*COS(Y3)/(Y3)**2 +
     &              2.0*SIN(Y3)/(Y3)**3)/2.0
         ELSE
            FY3=1.0
            FPY3=0.0
            SIG4=0.0
            SIG12=-1/6.0
         ENDIF

         U1=Y4*CTDY3 + Y5*STDY3 + Y6*P*FY3

         IF(ABS(U1).GT.EPS2)THEN
            GU1=1.0
         ELSE
            GU1=0.0
         ENDIF

         BETA=1-Y6*GU1*Y3
         IF(BETA.GT.(0.0))THEN
             ALPHA(K)=ALPHAMIN
         ELSE
             ALPHA(K)=ALPHAMAX
         ENDIF

         SIG1= -Y6*FY3*STD/2.0
         SIG2= Y6*FY3*CTD/2.0
         SIG3= Y4/2.0*STDY3 - Y5/2.0*CTDY3 + Y6*P*SIG4


         SIG6= CTDY3/2.0

         SIG8= STDY3/2.0
         SIG9= P*FY3/2.0
         SIG10 = -STD/2.0
         SIG11= CTD/2.0

c       Enter non-zero values:
```

```
      S(1,1)= -1 + H*CTDY3*SIG1
      S(1,2)= H*CTDY3*SIG2
      S(1,3)= H*(CTDY3*SIG3 + STDY3*U1/2.0)
      S(1,4)= H*(CTDY3*SIG6)
      S(1,5)= H*(CTDY3*SIG8)
      S(1,6)= H*CTDY3*SIG9
      S(1,7)= S(1,1) + 2.0
      S(1,8)= S(1,2)
      S(1,9)= S(1,3)
      S(1,10)= S(1,4)
      S(1,11)= S(1,5)
      S(1,12)= S(1,6)

      S(2,1)= H*STDY3*SIG1
      S(2,2)= -1 + H*STDY3*SIG2
      S(2,3)= H*(STDY3*SIG3 - CTDY3*U1/2.0)
      S(2,4)= H*(STDY3*SIG6)
      S(2,5)= H*(STDY3*SIG8)
      S(2,6)= H*STDY3*SIG9
      S(2,7)= S(2,1)
      S(2,8)= S(2,2) + 2.0
      S(2,9)= S(2,3)
      S(2,10)= S(2,4)
      S(2,11)= S(2,5)
      S(2,12)= S(2,6)

      S(3,1)= H*FY3*(P*SIG1+SIG10*U1)
      S(3,2)= H*FY3*(P*SIG2+SIG11*U1)
      S(3,3)= -1 + H*(ALPHA(K)*GU1/2.0 + P*(U1*SIG4 + SIG3*FY3))
      S(3,4)= H*P*FY3*SIG6
      S(3,5)= H*P*FY3*SIG8
      S(3,6)= H*P*FY3*SIG9
      S(3,7)= S(3,1)
      S(3,8)= S(3,2)
      S(3,9)= S(3,3) + 2.0
      S(3,10)= S(3,4)
      S(3,11)= S(3,5)
      S(3,12)= S(3,6)

      S(4,1)= H*Y6*STD*FY3*SIG1
      S(4,2)= H*Y6*STD*FY3*SIG2
      S(4,3)= H*Y6*STD*(U1*SIG4 + SIG3*FY3)
      S(4,4)= -1 + H*Y6*STD*FY3*SIG6
      S(4,5)= H*Y6*STD*FY3*SIG8
      S(4,6)= H*STD*FY3*(Y6*SIG9 + U1/2.0)
      S(4,7)= S(4,1)
      S(4,8)= S(4,2)
      S(4,9)= S(4,3)
      S(4,10)= S(4,4) + 2.0
      S(4,11)= S(4,5)
      S(4,12)= S(4,6)

      S(5,1)= -H*Y6*CTD*FY3*SIG1
      S(5,2)= -H*Y6*CTD*FY3*SIG2
      S(5,3)= -H*Y6*CTD*(U1*SIG4 + SIG3*FY3)
      S(5,4)= -H*Y6*CTD*SIG6*FY3
      S(5,5)= -1 - H*Y6*CTD*SIG8*FY3
      S(5,6)= -H*CTD*FY3*(Y6*SIG9 + U1/2.0)
      S(5,7)= S(5,1)
      S(5,8)= S(5,2)
      S(5,9)= S(5,3)
      S(5,10)= S(5,4)
      S(5,11)= S(5,5) + 2.0
      S(5,12)= S(5,6)

      S(6,1)= -H*(Y4*STDY3*SIG1 - Y5*CTDY3*SIG1 +
     &          Y6*FPY3*(P*SIG1+SIG10*U1))
      S(6,2)= -H*(Y4*STDY3*SIG2 - Y5*CTDY3*SIG2 +
     &          Y6*FPY3*(P*SIG2+SIG11*U1))
```

```
      S(6,3)= -H*(Y4*(-U1*CTDY3/2.0 + SIG3*STDY3)
     &            - Y5*(U1*STDY3/2.0 + CTDY3*SIG3)
     &           + Y6*P*(U1*SIG12 + SIG3*FPY3))
      S(6,4)= -H*(STDY3*(Y4*SIG6 + U1/2.0) - Y5*CTDY3*SIG6 +
     &            Y6*P*FPY3*SIG6)
      S(6,5)= -H*(Y4*STDY3*SIG8 - CTDY3*(Y5*SIG8 + U1/2.0) +
     &            Y6*P*FPY3*SIG8)
      S(6,6)= -1 -H*(Y4*STDY3*SIG9 - Y5*CTDY3*SIG9
     &                 +Y6*P*FPY3*SIG9
     &                 +(ALPHA(K)*GU1+P*U1*FPY3)/2.0)
      S(6,7)= S(6,1)
      S(6,8)= S(6,2)
      S(6,9)= S(6,3)
      S(6,10)= S(6,4)
      S(6,11)= S(6,5)
      S(6,12)= S(6,6) + 2.0

       S(1,JSF)= Y(1,K)-Y(1,K-1)+H*(CTDY3*U1)
       S(2,JSF)= Y(2,K)-Y(2,K-1)+H*(STDY3*U1)
       S(3,JSF)= Y(3,K)-Y(3,K-1)+H*(ALPHA(K)*GU1*Y3+P*U1*FY3)
       S(4,JSF)= Y(4,K)-Y(4,K-1)+H*Y6*U1*STD*FY3
       S(5,JSF)= Y(5,K)-Y(5,K-1)-H*Y6*U1*CTD*FY3
       S(6,JSF)= Y(6,K)-Y(6,K-1)-H*(Y4*U1*STDY3-Y5*U1*CTDY3+
     &            Y6*(ALPHA(K)*GU1+P*U1*FPY3))

          ENDIF

C     Dummy use of variables to prevent inocculous warning on MS Compiler
C     (Variables not used)
      IS1 = IS1
      ISF = ISF
      INDEXV(1) = INDEXV(1)
      NE = NE

      RETURN
      END
```

```
      SUBROUTINE XLATOR4(Y,YDOT,H,NYJ,NYK,X,THETAD,EPS,ALPHA,
     &                 X0,YO,THETA0,XF,YF,POS,EPS2)
****************************************************************
****************************************************************
*            MODIFIED 7/21/94 FOR FIXED VIRTUAL (TARGET) ROBOT PROBLEM
*      SUCH THAT UD=0 FOR ALL TIME

*      MODIFIED 7/25/94 FOR FUNCTION, G(U1) TIMES ALPHA TERM IN U2.
*                    (FEEDBACK REFINEMENT)
*      MODIFIED 7/26/94 FOR ALPHA AS INPUT, ALPHAMIN OR ALPHAMAX.
****************************************************************

       IMPLICIT REAL*8 (A-H, O-Z)

       PARAMETER(M=201)
       DIMENSION Y(NYJ,NYK), YDOT(5,M), X(M), POS(5,M),ALPHA(M),
     &           COST(M),NRG(M),U1TRAJ(M),GU1TRAJ(M)

       POS(1,1)=X0
       POS(2,1)=Y0
       POS(3,1)=THETA0
       POS(4,1)=XF
       POS(5,1)=YF

       NRG(1)=0

       DO 10 K=1,M

         IF(ABS(Y(3,K)).GT.EPS)THEN
           FY3= SIN(Y(3,K))/Y(3,K)
         ELSE
           FY3= 1.0
         ENDIF

         Pdel= Y(2,K)*COS(THETAD)-Y(1,K)*SIN(THETAD)
         U1= Y(4,K)*COS(THETAD-Y(3,K)) +
     &       Y(5,K)*SIN(THETAD-Y(3,K)) +
     &       Y(6,K)*Pdel*FY3

       U1TRAJ(K)=U1
*      ENERGY COST FUCTION FOR ROBOT5
       NRG(K)=(U1TRAJ(K)**2)/2 + ALPHA(K)

       IF(ABS(U1).GT.EPS2)THEN
          GU1=1.0
       ELSE
          GU1=0.0
       ENDIF

       GU1TRAJ(K)=GU1

         UD= 0.0

         YDOT(1,K)= COS(THETAD-Y(3,K))*U1
         YDOT(2,K)= SIN(THETAD-Y(3,K))*U1
         YDOT(3,K)= Pdel*U1*FY3 + ALPHA(k)*GU1*Y(3,K)
         YDOT(4,K)= COS(THETAD)*UD
         YDOT(5,K)= SIN(THETAD)*UD

       IF(K.GT.1)THEN
           POS(1,K)=XF-Y(1,K)
           POS(2,K)=YF-Y(2,K)
           POS(3,K)=THETAD-Y(3,K)
           POS(4,K)=XF
           POS(5,K)=YF
C
C      TRAPEZOIDAL INTEGRATION:
C
```

```
            COST(K)=COST(K-1)+H*(NRG(K)+NRG(K-1))/2
          ENDIF

        WRITE(34,80) X(K),YDOT(1,K),YDOT(2,K),YDOT(3,K),YDOT(4,K),
     &               YDOT(5,K)
        WRITE(35,80) X(K),POS(1,K),POS(2,K),POS(3,K),POS(4,K),
     &               POS(5,K)
        WRITE(37,81) X(K),U1TRAJ(K),GU1TRAJ(K),COST(K)

10      CONTINUE

80      FORMAT(2X,6F20.10)
81      FORMAT(2X,4F20.10)

      RETURN
      END
```

APPENDIX F

Program Files Specific to Robot 5

ROBOT5.FOR

DIFEQ.FOR (for Robot 5)

XLATOR5.FOR

```
      PROGRAM ROBOT5
*************************************************************************
*     SINGLE ROBOT WITH TERMINAL COST, g(U1), AND PROPORTIONAL ALPHA CONTROL
*************************************************************************
*     Source for subroutines Red, Pinvs, Solvde, and Bksub and model for
*     Difeq and Diskmain:
*          Numerical Recipes, William H. Press, et al
*************************************************************************

      IMPLICIT REAL*8 (A-H, O-Z)

      PARAMETER(NE=6,M=201,NB=3,NCI=NE,NCJ=NE-NB+1,NCK=M+1,NSI=NE,
     &    NSJ=2*NE+1, NYJ=NE, NYK=M)

      DIMENSION SCALV(NE),INDEXV(NE),Y(NE,M),C(NCI,NCJ,NCK),S(NSI,NSJ),
     &          YDOT(NE-1,M),POS(NE-1,M),U1TRAJ(M)
      COMMON X(M), H, DELX0, DELY0, DELTHET0, THETAD, EPS, CWT,ALPHA(M),
     &      EPS2, ALPHAMIN

*************************************************************************
*     Variable description:
*
*     GENERAL PROGRAM VARIABLES:
*-------------------------------------------------------------------------
*     NE:    Number of independent equations describing system
*     M:     Number of Meshpoints, divisions of independent variable, time
*     NB:    Number of Boundary Conditions known at initial condition
*     C:     3-D Array for storage of corrections for each iteration
*               Note: largest array in program
*     NCI, NCJ, NCK:
*            dimension variables of C array, must satisfy equations
*            found in parameter statement
*     S:     array for storage of blocks of solution of Difeq.
*     NSI, NSJ:
*            dimension variables of 2-D S array, must satisfy equations
*            found in parameter statement
*     Y:     2-D array containing initial guess for each point.  This array
*            is updated by calculated corrections.  When the corrections
*            are sufficiently small, convergence is acheived.
*     X:     Array for independent variable, time. Used only for comparison
*            of dependent variables after program completes.
*     SCALV: Array of values representing the typical magnitude of the
*            dependent variables.  Used for controlling correction magnitude.
*     INDEXV:lists column ordering for variables in S array, not used in this
*            program
*     ITMAX: Maximum number of iterations
*     CONV:  Convergence criteria for corrections to Y
*     SLOWC: Controls fraction of corrections applied to Y
*         H: Increment of independent variable, divisions between mesh points
*
*     PROBLEM SPECIFIC VARIABLES:
*-------------------------------------------------------------------------
*
*       X0:            Initial X coordinate of robot
*       Y0:            Initial Y coordinate of robot
*       THETA0:               Initial angle wrt X axis of robot
*       XF:            Desired final X coordinate of robot
*       YF:            Desired final Y coordinate of robot
*       THETAD:               Desired final angle coordinate of robot
*       DELX0:         Initial boundary condition for state variable delta-X
*       DELY0:         Initial boundary condition for state variable delta-Y
*       DELTHET0:      Initial boundary condition for state variable delta-theta
*       EPS:           Smallest value for which f(delta-x)=Sin(delta-x)/(delta-x)
*       EPS2:          Smallest value of U1 for which g(U1)=1
*       ALPHA:         Rotational gain related to delta-theta-dot
*       ALPHAMAX:      DUMMY VARIABLE USED FOR CONTINUITY WITH OTHER FORMS OF
ROBOT.
```

```
*       CWT:            Weighting parameter for Terminal Costs
*       RL1,RL2,RL3:    Initial guess amplitude for lambda costates.
*       OHM1,OHM2,OHM3:         Initial guess frequencies for lambda costates.
*       DC1,DC2,DC3:    Initial guess d.c. offsets for lambda costates.
*       PHI1,PHI2,PHI3:         Initial guess phase lag for lambda costates.
*       SL1,SL2,SL3:    Initial guess SCALV, scale sizes, for lambda costates.
*       POS(NE-1,M):    Position Trajectory for x,y,theta, xd and yd.
*
************************************************************************

      OPEN(21,FILE='xplot.rob5', STATUS='UNKNOWN')
      OPEN(22,FILE='lplot.rob5', STATUS='UNKNOWN')
      OPEN(30,FILE='xiplot.rob5', STATUS='UNKNOWN')
      OPEN(31,FILE='liplot.rob5', STATUS='UNKNOWN')
      OPEN(32,FILE='itplot.rob5', STATUS='UNKNOWN')
      OPEN(33,FILE='testdat.rob5', STATUS='UNKNOWN')
      OPEN(34,FILE='dotplot.rob5', STATUS='UNKNOWN')
      OPEN(35,FILE='posplot.rob5', STATUS='UNKNOWN')
      OPEN(36,FILE='alplot.rob5', STATUS='UNKNOWN')
      OPEN(37,FILE='ulplot.rob5', STATUS='UNKNOWN')

      PI= 3.141592654

      PRINT *,'ENTER ITMAX, CONV, SLOWC'
      READ *, ITMAX, CONV, SLOWC
      PRINT *,'ENTER ALPHAMIN'
      READ *, ALPHA MIN
       print *,'dummy read for compatibility , enter #'
       READ *,DUMMY
      PRINT *,'ENTER C, WEIGHTING PARAMTER FOR TERMINAL COST'
      READ *, CWT
      PRINT *,'ENTER EPSILON for f(deltheta)'
      READ *, EPS
      PRINT *,'ENTER EPSILON 2 for g(U1)'
      READ *, EPS2
      PRINT *, 'ENTER INITIAL X,Y, AND THETA(degrees)'
      READ *, X0,Y0,THETA0
      THETA0=THETA0*PI/180
      PRINT *, 'ENTER FINAL X,Y, AND THETA(degrees)'
      READ *, XF,YF,THETAD
      THETAD=THETAD*PI/180
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA AMPLITUDES'
      READ *, RL1,RL2,RL3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA FREQUENCIES'
      READ *, OHM1,OHM2,OHM3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA DC OFFSETS'
      READ *, DC1,DC2,DC3
      PRINT *,'ENTER INITIAL GUESS FOR 3 LAMBDA PHASES'
      READ *, PHI1, PHI2, PHI3
       PHI1=PHI1*PI/180
       PHI2=PHI2*PI/180
       PHI3=PHI3*PI/180
      PRINT *,'ENTER INITIAL GUESS FOR SIZE OF 3 LAMBDA VALUES'
      READ *, SL1,SL2,SL3

      DELX0=(XF-X0)
      DELY0=(YF-Y0)
      DELTHET0=(THETAD-THETA0)

      H=1./(M-1)

C     NO INDEX CHANGES NECESSARY
C     Index Scale used by SOLVDE.FOR

      INDEXV(1)=1
      INDEXV(2)=2
      INDEXV(3)=3
      INDEXV(4)=4
      INDEXV(5)=5
```

```
      INDEXV(6)=6

C     INITIAL GUESS FOR ALL POINTS, 1 - M

C     Initialize independent vector X (time)

      DO 11 K=1,M
         X(K)=(K-1)*H
 11   CONTINUE

C      INITIAL GUESS:

C     Enter initial values for all meshpoints
C       NOTE: BOUNDARY CONDITONS FOR Y(1)-Y(3) ARE
C             ENTERED AT POINTS 1 AND M DURING THE INITIAL GUESS!!!
C             THESE NUMBERS MUST COINCIDE WITH ANY DESIRED B.C.!!!

       WRITE(33,*) 'STRAIGHT LINE GUESS'
       DO 12 K=1,M
         y(1,K)=DELX0-(DELX0*(K-1)/(M-1))
         y(2,K)=DELY0-(DELY0*(K-1)/(M-1))
         y(3,K)=DELTHET0-(DELTHET0*(K-1)/(M-1))
         y(4,K)=RL1*SIN(2*PI*OHM1*(K-1)/(M-1)+PHI1)+DC1
         y(5,K)=RL2*SIN(2*PI*OHM2*(K-1)/(M-1)+PHI2)+DC2
         y(6,K)=RL3*SIN(2*PI*OHM3*(K-1)/(M-1)+PHI3)+DC3
 12    CONTINUE

*       INITIAL ALPHA DETERMINATION:
*********************************************************************
        IF(Y(3,1).GT.EPS) THEN
          FY3= SIN(Y(3,1))/(Y(3,1))
        ELSE
          FY3= 1.0
        ENDIF

        U10=Y(4,1)*COS(THETAD-Y(3,1)) + Y(5,1)*SIN(THETAD-Y(3,1))
     &    + Y(6,1)*(Y(2,1)*COS(THETAD)-Y(1,1)*SIN(THETAD))*FY3

        IF(ABS(U10).GT.EPS2)THEN
          GU1=1.0
        ELSE
          GU1=0.0
        ENDIF

        BETA=Y(6,1)*GU1*Y(3,1)

        IF(BETA.LT.ALPHAMIN)THEN
           ALPHA(1)=ALPHAMIN
        ELSE
           ALPHA(1)=BETA
        ENDIF
*********************************************************************

C     Write initial guess to file

      DO 13 K=1,M
           WRITE(30,80) X(K),Y(1,K),Y(2,K),Y(3,K)
           WRITE(31,80) X(K),Y(4,K),Y(5,K),Y(6,K)
 13        CONTINUE

C     Scalv set to approximate size of typical values of known solution

      SCALV(1)=ABS(DELX0/2)+.01
      SCALV(2)=ABS(DELY0/2)+.01
      SCALV(3)=ABS(2*DELTHET0/M)+.01
      SCALV(4)=ABS(SL1)+.01
      SCALV(5)=ABS(SL2)+.01
      SCALV(6)=ABS(SL3)+.01
```

```
C     WRITE TEST DATA TO FILE
      WRITE(33,*) 'ITMAX =',ITMAX
      WRITE(33,*) 'CONV  =',CONV
      WRITE(33,*) 'SLOWC =',SLOWC
      WRITE(33,*) 'ALPHA MIN =',ALPHAMIN
      WRITE(33,*) 'CWT =',CWT
      WRITE(33,*) 'EPSILON =',EPS
      WRITE(33,*) 'EPSILON 2 =',EPS2
      WRITE(33,*) 'INITIAL X,Y & THETA = ', X0,Y0,(THETA0*180/PI)
      WRITE(33,*) 'FINAL   X,Y & THETA = ', XF,YF,(THETAD*180/PI)
      WRITE(33,*) 'INITIAL LAMBDA VALUE AMPLITUDE', RL1,RL2,RL3
      WRITE(33,*) 'SIZE OF LAMBDA VALUES', SL1,SL2,SL3
      WRITE(33,*) 'DELX0 =', DELX0
      WRITE(33,*) 'DELY0 =', DELY0
      WRITE(33,*) 'DELTHETO =', DELTHETO

      CALL SOLVDE(ITMAX,CONV,SLOWC,SCALV,INDEXV,NE,NB,M,Y,NYJ,NYK,
     *     C,NCI,NCJ,NCK,S,NSI,NSJ)

C     Write final Y values to file:

        DO 181 k = 1,M
          WRITE(21,80) X(K),Y(1,k),Y(2,k),Y(3,k)
          WRITE(22,80) X(K),Y(4,k),Y(5,k),Y(6,k)
         WRITE(36,81) X(K),ALPHA(K)
 181    CONTINUE

 80   FORMAT(2X,4F15.4)
 81   FORMAT(2X,2F15.4)

      CALL XLATOR5(Y,YDOT,H,NYJ,NYK,X,THETAD,EPS,ALPHA,
     &                 X0,Y0,THETA0,XF,YF,POS,EPS2,U1TRAJ)

       PRINT *,'PROGRAM COMPLETED'

      CLOSE(21)
      CLOSE(22)
      CLOSE(30)
      CLOSE(31)
      CLOSE(32)
      CLOSE(33)
      CLOSE(34)
      CLOSE(35)
      CLOSE(36)
      CLOSE(37)

      END
```

```
      SUBROUTINE DIFEQ(K,K1,K2,JSF,IS1,ISF,INDEXV,NE,S,NSI,NSJ,Y,NYJ,
     *                 NYK)
      IMPLICIT REAL*8 (A-H, O-Z)

**********************************************************************
* MODIFIED 7/18/94 TO INCLUDE TERMINAL COST AT FINA BOUNDARY CONDITION
* MODIFIED 7/21/94 FOR NON-MOVING VIRTUAL (TARGET) ROBOT
* MODIFIED 7/25/94 FOR FUNCTION, G(U1) TIMES ALPHA TERM IN U2.
*                  (FEEDBACK REFINEMENT)
* MODIFIED 7/26/94 FOR ALPHA AS INPUT
* MODIFIED 7/26/94 FOR PROPORTIONAL ALPHA AS INPUT
**********************************************************************
      PARAMETER(M=201)
      DIMENSION Y(NYJ,NYK), S(NSI,NSJ), INDEXV(NYJ)
      COMMON X(M), H, DELX0, DELY0, DELTHETO, THETAD, EPS, CWT,ALPHA(M),
     &       EPS2, ALPHAMIN

C     Initialize matrix S as 0

      DO 10 I=1,NSI
          DO 9 J=1,NSJ
             S(I,J) = 0.0
 9        CONTINUE
 10   CONTINUE

**********************************************************************
C     Initial Boundary Conditions

      IF(K.EQ.K1) THEN

C     Enter non-zero values:

          DO 11 I= 1,3
             S(3+I,6+I)=1.0
 11       CONTINUE

C    Initial values in right hand vector for initial block

          S(4,JSF)= y(1,1)-DELX0
          S(5,JSF)= y(2,1)-DELY0
          S(6,JSF)= y(3,1)-DELTHETO

**********************************************************************
C     End Boundary Conditions

      ELSE IF (K.GT.K2) THEN

C     Enter non-zero values:

          S(1,7)= CWT
          S(1,8)= 0.0
          S(1,9)= 0.0
          S(1,10)= -1.0
          S(1,11)= 0.0
          S(1,12)= 0.0

          S(2,7)= 0.0
          S(2,8)= CWT
          S(2,9)= 0.0
          S(2,10)= 0.0
          S(2,11)= -1.0
          S(2,12)= 0.0

          S(3,7)= 0.0
          S(3,8)= 0.0
          S(3,9)= CWT
          S(3,10)= 0.0
          S(3,11)= 0.0
          S(3,12)= -1.0
```

```
C     Final values in right hand vector for final block

            S(1,JSF)= Y(1,M)*CWT - Y(4,M)
            S(2,JSF)= Y(2,M)*CWT - Y(5,M)
            S(3,JSF)= Y(3,M)*CWT - Y(6,M)

**********************************************************************
C      Interior Points
C          Derived from Finite Difference Equations of Motion

       ELSE

C      Pre-calculation of commonly used variables:

       Y1=(Y(1,K)+Y(1,K-1))/2.0
       Y2=(Y(2,K)+Y(2,K-1))/2.0
       Y3=(Y(3,K)+Y(3,K-1))/2.0
       Y4=(Y(4,K)+Y(4,K-1))/2.0
       Y5=(Y(5,K)+Y(5,K-1))/2.0
       Y6=(Y(6,K)+Y(6,K-1))/2.0

         CTD=COS(THETAD)
         STD=SIN(THETAD)
         CTDY3=COS(THETAD-Y3)
         STDY3=SIN(THETAD-Y3)

          P= Y2*CTD-Y1*STD

         IF(ABS(Y3).GT.EPS)THEN
            FY3=SIN(Y3)/(Y3)
            FPY3=(COS(Y3)/Y3)-(SIN(Y3)/(Y3**2))
            SIG4=FPY3/2.0
             SIG12=(-SIN(Y3)/Y3-2.0*COS(Y3)/(Y3)**2 +
     &            2.0*SIN(Y3)/(Y3)**3)/2.0
         ELSE
            FY3=1.0
            FPY3=0.0
            SIG4=0.0
            SIG12=-1/6.0
         ENDIF

         U1=Y4*CTDY3 + Y5*STDY3 + Y6*P*FY3

         IF(ABS(U1).GT.EPS2)THEN
            GU1=1.0
         ELSE
            GU1=0.0
         ENDIF

         BETA=Y6*GU1*Y3
         IF(BETA.LT.ALPHAMIN)THEN
             ALPHA(K)=ALPHAMIN
             DELAL3=0.0
             DELAL6=0.0
         ELSE
             ALPHA(K)=BETA
             DELAL3=Y6/2
             DELAL6=Y3/2
         ENDIF

         SIG1= -Y6*FY3*STD/2.0
         SIG2= Y6*FY3*CTD/2.0
         SIG3= Y4/2.0*STDY3 - Y5/2.0*CTDY3 + Y6*P*SIG4


         SIG6= CTDY3/2.0

         SIG8= STDY3/2.0
         SIG9= P*FY3/2.0
```

```
      SIG10 = -STD/2.0
      SIG11= CTD/2.0

c     Enter non-zero values:

      S(1,1)= -1 + H*CTDY3*SIG1
      S(1,2)= H*CTDY3*SIG2
      S(1,3)= H*(CTDY3*SIG3 + STDY3*U1/2.0)
      S(1,4)= H*(CTDY3*SIG6)
      S(1,5)= H*(CTDY3*SIG8)
      S(1,6)= H*CTDY3*SIG9
      S(1,7)= S(1,1) + 2.0
      S(1,8)= S(1,2)
      S(1,9)= S(1,3)
      S(1,10)= S(1,4)
      S(1,11)= S(1,5)
      S(1,12)= S(1,6)

      S(2,1)= H*STDY3*SIG1
      S(2,2)= -1 + H*STDY3*SIG2
      S(2,3)= H*(STDY3*SIG3 - CTDY3*U1/2.0)
      S(2,4)= H*(STDY3*SIG6)
      S(2,5)= H*(STDY3*SIG8)
      S(2,6)= H*STDY3*SIG9
      S(2,7)= S(2,1)
      S(2,8)= S(2,2) + 2.0
      S(2,9)= S(2,3)
      S(2,10)= S(2,4)
      S(2,11)= S(2,5)
      S(2,12)= S(2,6)

      S(3,1)= H*FY3*(P*SIG1+SIG10*U1)
      S(3,2)= H*FY3*(P*SIG2+SIG11*U1)
      S(3,3)= -1 + H*(GU1*(ALPHA(K)/2.0 + DELAL3*Y3)
     &           + P*(U1*SIG4 + SIG3*FY3))
      S(3,4)= H*P*FY3*SIG6
      S(3,5)= H*P*FY3*SIG8
      S(3,6)= H*(P*FY3*SIG9+Y3*GU1*DELAL6)
      S(3,7)= S(3,1)
      S(3,8)= S(3,2)
      S(3,9)= S(3,3) + 2.0
      S(3,10)= S(3,4)
      S(3,11)= S(3,5)
      S(3,12)= S(3,6)

      S(4,1)= H*Y6*STD*FY3*SIG1
      S(4,2)= H*Y6*STD*FY3*SIG2
      S(4,3)= H*Y6*STD*(U1*SIG4 + SIG3*FY3)
      S(4,4)= -1 + H*Y6*STD*FY3*SIG6
      S(4,5)= H*Y6*STD*FY3*SIG8
      S(4,6)= H*STD*FY3*(Y6*SIG9 + U1/2.0)
      S(4,7)= S(4,1)
      S(4,8)= S(4,2)
      S(4,9)= S(4,3)
      S(4,10)= S(4,4) + 2.0
      S(4,11)= S(4,5)
      S(4,12)= S(4,6)

      S(5,1)= -H*Y6*CTD*FY3*SIG1
      S(5,2)= -H*Y6*CTD*FY3*SIG2
      S(5,3)= -H*Y6*CTD*(U1*SIG4 + SIG3*FY3)
      S(5,4)= -H*Y6*CTD*SIG6*FY3
      S(5,5)= -1 - H*Y6*CTD*SIG8*FY3
      S(5,6)= -H*CTD*FY3*(Y6*SIG9 + U1/2.0)
      S(5,7)= S(5,1)
      S(5,8)= S(5,2)
      S(5,9)= S(5,3)
      S(5,10)= S(5,4)
      S(5,11)= S(5,5) + 2.0
```

```
      S(5,12)= S(5,6)

      S(6,1)= -H*(Y4*STDY3*SIG1 - Y5*CTDY3*SIG1 +
     &              Y6*FPY3*(P*SIG1+SIG10*U1))
      S(6,2)= -H*(Y4*STDY3*SIG2 - Y5*CTDY3*SIG2 +
     &              Y6*FPY3*(P*SIG2+SIG11*U1))
      S(6,3)= -H*(Y4*(-U1*CTDY3/2.0 + SIG3*STDY3)
     &            - Y5*(U1*STDY3/2.0 + CTDY3*SIG3)
     &            + Y6*P*(U1*SIG12 + SIG3*FPY3 + DELAL3*GU1))
      S(6,4)= -H*(STDY3*(Y4*SIG6 + U1/2.0) - Y5*CTDY3*SIG6 +
     &             Y6*P*FPY3*SIG6)
      S(6,5)= -H*(Y4*STDY3*SIG8 - CTDY3*(Y5*SIG8 + U1/2.0) +
     &             Y6*P*FPY3*SIG8)
      S(6,6)= -1 -H*(Y4*STDY3*SIG9 - Y5*CTDY3*SIG9
     &                 +Y6*(P*FPY3*SIG9 + DELAL6*GU1)
     &                 +(ALPHA(K)*GU1+P*U1*FPY3)/2.0)
      S(6,7)= S(6,1)
      S(6,8)= S(6,2)
      S(6,9)= S(6,3)
      S(6,10)= S(6,4)
      S(6,11)= S(6,5)
      S(6,12)= S(6,6) + 2.0

      S(1,JSF)= Y(1,K)-Y(1,K-1)+H*(CTDY3*U1)
      S(2,JSF)= Y(2,K)-Y(2,K-1)+H*(STDY3*U1)
      S(3,JSF)= Y(3,K)-Y(3,K-1)+H*(ALPHA(K)*GU1*Y3+P*U1*FY3)
      S(4,JSF)= Y(4,K)-Y(4,K-1)+H*Y6*U1*STD*FY3
      S(5,JSF)= Y(5,K)-Y(5,K-1)-H*Y6*U1*CTD*FY3
      S(6,JSF)= Y(6,K)-Y(6,K-1)-H*(Y4*U1*STDY3-Y5*U1*CTDY3+
     &             Y6*(ALPHA(K)*GU1+P*U1*FPY3))

         ENDIF

C     Dummy use of variables to prevent inocculous warning on MS Compiler
C     (Variables not used)
      IS1 = IS1
      ISF = ISF
      INDEXV(1) = INDEXV(1)
      NE = NE

      RETURN
      END
```

```
      SUBROUTINE XLATOR5(Y,YDOT,H,NYJ,NYK,X,THETAD,EPS,ALPHA,
     &                   X0,YO,THETA0,XF,YF,POS,EPS2,U1TRAJ)

*****************************************************************
*****************************************************************
*       MODIFIED 7/21/94 FOR FIXED VIRTUAL (TARGET) ROBOT PROBLEM
*       SUCH THAT UD=0 FOR ALL TIME
*       MODIFIED 7/25/94 FOR FUNCTION, G(U1) TIMES ALPHA TERM IN U2.
*                      (FEEDBACK REFINEMENT)
*       MODIFIED 7/26/94 FOR PROPRTIONAL ALPHA CONTROL
*****************************************************************

        IMPLICIT REAL*8 (A-H, O-Z)

        PARAMETER(M=201)
        DIMENSION Y(NYJ,NYK), YDOT(5,M), X(M), POS(5,M),ALPHA(M),
     &            COST(M),NRG(M),U1TRAJ(M),GU1TRAJ(M)
        POS(1,1)=X0
        POS(2,1)=Y0
        POS(3,1)=THETA0
        POS(4,1)=XF
        POS(5,1)=YF

        NRG(1)=0

        DO 10 K=1,M

          IF(ABS(Y(3,K)).GT.EPS)THEN
            FY3= SIN(Y(3,K))/Y(3,K)
          ELSE
            FY3= 1.0
          ENDIF

          Pdel= Y(2,K)*COS(THETAD)-Y(1,K)*SIN(THETAD)
          U1= Y(4,K)*COS(THETAD-Y(3,K)) +
     &        Y(5,K)*SIN(THETAD-Y(3,K)) +
     &        Y(6,K)*Pdel*FY3

        U1TRAJ(K)=U1
*       COST FUNCTION FOR ROBOT6 ONLY
*       NRG(K)=(U1TRAJ(K)**2 + ALPHA(K)**2)/2
*       COST FUNCTION IN SAME FORM AS ROBOT5 FOR COMPARISON:
        NRG(K)=(U1TRAJ(K)**2)/2 + ALPHA(K)

        IF(ABS(U1).GT.EPS2)THEN
           GU1=1.0
        ELSE
           GU1=0.0
        ENDIF

        GU1TRAJ(K)=GU1

          UD= 0.0

          YDOT(1,K)= COS(THETAD-Y(3,K))*U1
          YDOT(2,K)= SIN(THETAD-Y(3,K))*U1
          YDOT(3,K)= Pdel*U1*FY3 + ALPHA(K)*GU1*Y(3,K)
          YDOT(4,K)= COS(THETAD)*UD
          YDOT(5,K)= SIN(THETAD)*UD

          IF(K.GT.1)THEN
            POS(1,K)=XF-Y(1,K)
            POS(2,K)=YF-Y(2,K)
            POS(3,K)=THETAD-Y(3,K)
            POS(4,K)=XF
            POS(5,K)=YF
C
C        TRAPEZOIDAL INTEGRATION:
C
```

```
            COST(K)=COST(K-1)+H*(NRG(K)+NRG(K-1))/2
          ENDIF

        WRITE(34,80) X(K),YDOT(1,K),YDOT(2,K),YDOT(3,K),YDOT(4,K),
     &               YDOT(5,K)
        WRITE(35,80) X(K),POS(1,K),POS(2,K),POS(3,K),POS(4,K),
     &               POS(5,K)
        WRITE(37,81) X(K),U1TRAJ(K),GU1TRAJ(K),COST(K)

10      CONTINUE

80      FORMAT(2X,6F20.10)
81      FORMAT(2X,4F20.10)

      RETURN
      END
```

APPENDIX G

Program Subroutines Used by All Programs

SOLVDE.FOR

PINVS.FOR

RED.FOR

BKSUB.FOR

```
      subroutine solvde(itmax,conv,slowc,scalv,indexv,ne,nb,m,
     1                  y,nyj,nyk,c,nci,ncj,nck,s,nsi,nsj)
C
      implicit real*8 (a-h, o-z)
C
      PARAMETER (NMAX=8)
      dimension y(nyj,nyk),c(nci,ncj,nck),s(nsi,nsj)
      dimension scalv(nyj),indexv(nyj)
      dimension ermax(NMAX),kmax(NMAX)

C
      k1 = 1
      k2 = m
      nvars = ne*m
      j1 = 1
      j2 = nb
      j3 = nb + 1
      j4 = ne
      j5 = j4 + j1
      j6 = j4 + j2
      j7 = j4 + j3
      j8 = j4 + j4
      j9 = j8 + j1
      ic1 = 1
      ic2 = ne - nb
      ic3 = ic2 + 1
      ic4 = ne
      jc1 = 1
      jcf = ic3
      do 16 it = 1,itmax
      k = k1
      call difeq(k,k1,k2,j9,ic3,ic4,indexv,ne,s,nsi,nsj,y,nyj,nyk)
      call pinvs(ic3,ic4,j5,j9,jc1,k1,c,nci,ncj,nck,s,nsi,nsj)
      do 11 k = k1+1,k2
      kp = k - 1
      call difeq(k,k1,k2,j9,ic1,ic4,indexv,ne,s,nsi,nsj,y,nyj,nyk)
      call red(ic1,ic4,j1,j2,j3,j4,j9,ic3,jc1,jcf,kp,c,nci,ncj,nck,
     &        s,nsi,nsj)
          call pinvs(ic1,ic4,j3,j9,jc1,k,c,nci,ncj,nck,s,nsi,nsj)

 11   continue

      k = k2 + 1
      call difeq(k,k1,k2,j9,ic1,ic2,indexv,ne,s,nsi,nsj,y,nyj,nyk)

      call red(ic1,ic2,j5,j6,j7,j8,j9,ic3,jc1,jcf,k2,c,nci,ncj,nck,
     1        s,nsi,nsj)

      call pinvs(ic1,ic2,j7,j9,jcf,k2+1,c,nci,ncj,nck,s,nsi,nsj)

      call bksub(ne,nb,jcf,k1,k2,c,nci,ncj,nck)

      err = 0.0
      do 13 j = 1,ne
      jv = indexv(j)
      errj = 0.0
      km = 0.0
      vmax = 0.0
      do 12 k = k1,k2
      vz = abs(c(j,1,k))
      if(vz.gt.vmax) then
      vmax = vz
      km = k
      endif
      errj = errj + vz
 12   continue
      err = err + errj/scalv(jv)
```

```
      ermax(j) = c(j,1,km)/scalv(jv)
      kmax(j) = km
13     continue
      err = err/nvars
      fac = slowc/max(slowc,err)
      do 15 jv = 1,ne
      j = indexv(jv)
      do 14 k = k1,k2
      y(j,k) = y(j,k) - fac*c(jv,1,k)
14     continue
15     continue
ccc      WRITE(*,101) IT,ERR,Y(5,1),Y(6,1),Y(7,1),Y(8,1)
ccc       WRITE(32,101) IT,ERR,Y(1,100),Y(2,100),Y(3,100),Y(4,100),
ccc   &                 Y(5,100),Y(6,100),Y(7,100),Y(8,100)
ccc       write(*,100) it,err,fac,(kmax(j),ermax(j),j=1,ne)
       WRITE(*,80) IT, ERR
       WRITE(32,80) IT, ERR
80    FORMAT(2X,I10,F20.8)
      if(err.lt.conv) then
              write(33,*) 'last iteration: ',it
              return
      endif
16     continue
       WRITE(33,*) 'itmax exceeded'
101    format(1X,I10,1X,9f15.6)
100     format(1x,i4,2f12.6,(/5x,i5,f12.6))
      return
      end
```

```fortran
      subroutine pinvs(ie1,ie2,je1,jsf,jc1,k,c,nci,ncj,nck,s,nsi,nsj)
c
       implicit real*8 (a-h, o-z)
c
       PARAMETER (ZERO=0., ONE=1., NMAX=8)
      dimension c(nci,ncj,nck), s(nsi,nsj)
       dimension pscl(NMAX),indxr(NMAX)
c
c
       je2 = je1+ie2-ie1
      js1 = je2+1
      do 12 i = ie1,ie2
      big = zero
      do 11 j = je1,je2
      if(abs(s(i,j)).gt.big) big = abs(s(i,j))
 11     continue
      if(big.eq.zero) pause 'Singular matrix, rows all 0'
      pscl(i) = one/big
      indxr(i) = 0
 12     continue
      do 18 id = ie1,ie2
      piv = zero
      do 14 i = ie1,ie2
      if(indxr(i).eq.0) then
      big = zero
      do 13 j = je1,je2
      if(abs(s(i,j)).gt.big) then
      jp = j
      big = abs(s(i,j))
      endif
 13     continue
      if(big*pscl(i).gt.piv) then
      ipiv = i
      jpiv = jp
      piv = big*pscl(i)
      endif
      endif
 14     continue
      if(s(ipiv,jpiv).eq.zero) pause 'Singular matrix'
      indxr(ipiv) = jpiv
      pivinv = one/s(ipiv,jpiv)
      do 15 j =je1,jsf
      s(ipiv,j) = s(ipiv,j)*pivinv
 15     continue
      s(ipiv,jpiv) = one
      do 17 i =ie1,ie2
      if(indxr(i).ne.jpiv) then
      if(s(i,jpiv).ne.zero) then
      dum = s(i,jpiv)
      do 16 j =je1,jsf
      s(i,j) = s(i,j) - dum*s(ipiv,j)
 16     continue
      s(i,jpiv) = zero
      endif
      endif
 17     continue
 18     continue
      jcoff = jc1-js1
      icoff = ie1-je1
      do 21 i=ie1,ie2
      irow = indxr(i) + icoff
      do 19 j = js1,jsf
       c(irow,j+jcoff,k) = s(i,j)
 19     continue
 21     continue
       return
      end
```

```
      subroutine red(iz1,iz2,jz1,jz2,jm1,jm2,jmf,ic1,jc1,jcf,kc,
     1               c,nci,ncj,nck,s,nsi,nsj)
c
       implicit real*8 (a-h, o-z)
c
       dimension c(nci,ncj,nck), s(nsi,nsj)

      loff = jc1-jm1
      ic = ic1
      do 14 j = jz1,jz2
      do 12 l = jm1,jm2
      vx = c(ic,l+loff,kc)
      do 11 i = iz1,iz2
      s(i,l) = s(i,l) - s(i,j)*vx
 11     continue
 12     continue
      vx = c(ic,jcf,kc)
      do 13 i = iz1,iz2
      s(i,jmf) = s(i,jmf) - s(i,j)*vx
 13     continue
      ic = ic + 1
 14     continue
       return
      end
```

```
      subroutine bksub(ne,nb,jf,k1,k2,c,nci,ncj,nck)
c
       implicit real*8 (a-h, o-z)
c
      dimension c(nci,ncj,nck)
      nbf = ne-nb
      im = 1
      do 13 k = k2, k1, -1
      if(k.eq.k1) im = nbf+1
      kp = k + 1
      do 12 j = 1,nbf
      xx = c(j,jf,kp)
      do 11 i = im,ne
      c(i,jf,k) = c(i,jf,k) - c(i,j,k)*xx
 11     continue
 12     continue
 13     continue
      do 16 k = k1,k2
      kp = k + 1
      do 14 i = 1,nb
      c(I,1,k) = c(i+nbf,jf,k)
 14     continue
      do 15 i = 1,nbf
      c(i+nb,1,k) = c(i,jf,kp)
 15     continue
 16     continue
      return
      end
```

REFERENCES

1. Pomet, J., "Explicit Design of Time Varying Stablilizing Control Laws For a Class of Controllable Systems Without Drift", *Systems & Control Letters* No. 18, pp 147-158, Elsevier Science Publishers, 1992

2. Canudas de Wit, C., H. Khennouf, C. Sampson, "Nonlinear Control Design For Mobile Robots", pp. 2-36, Technical Paper No. 93 075, 28 July 1993

3. Canuda de Wit, C., O.J. Sørdalen, "Exponential Stabilization of Mobile Robots with Nonholonomic Constraints", *IEEE Transactions on Automatic Control*, Vol 37, No. 11, November 1992

4. Greenwood, D.T., *Principles of Dynamics*, 2nd Ed.,p 244-246, Prentice Hall Inc. 1988

5. Junkins, J.L., *Optimal Spacecraft Rotational Maneuvers*, pp. 180-183, Elsevier Science Publishing Co. Inc., 1986

6. Press, W.H. et al, *Numerical Recipes*, pp. 578-615, Cambridge University Press, 1989

7. Cameron, J.M., *Modeling and Motion Planning for Non-Holonomic Systems*, Chapter 5, Ph.D. Thesis, Georgia Institute of Technology, December 1993

8. Brockett, R.W., Assymptotic Stability and Feedback Stabilizations in R.W. Brockett, R.S. Millman & H.J. Sussman, *Differential Geometric Control Theory*, pp. 181-208, Birkhauser,1983

9. M. Vidyasagar, *Nonlinear Systems Analysis*, 2nd Ed., p. 165, Prentice Hall, 1989

10. D. Chen, R. Mukherjee, "Assymptotic Stability Theorem for Autonomous Systems", *Journal of Guidance , Control, and Dynamics*, Vol 16, pp. 961-963, September-October 1993

11. Y. Nakamura, R. Mukherjee, "Nonholonomic Path Planning of Space Robots via a Bidirectional Approach", *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 4, pp. 500-514, August 1994

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Techical Information Center Cameron Station Alexandria, Virgina 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code ME Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4.	Curriculum Officer, Code 34 Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5000	1
5.	Commandant, (G-MTH-2) United States Coast Guard 2100 Second St. S.W. Washington, D.C. 20593-0001	2
6.	Dr. Ranjan Mukherjee, Code ME/MK Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5000	2
7.	LT Bryan R. Emond, USCG U.S. Coast Guard Marine Safety Center 400 7th St. S.W. Washington, DC 20590-0001	1