SOLUTION METHODS FOR NONLINEAR OPTIMAL CONTROL OF DISTRIBUTED ROBOTIC SYSTEMS

An Undergraduate Thesis

by

Andrés K. Valenzuela,

Dr. Bill Goodwine, Director

Undegraduate Program in Aerospace and Mechanical Engineering

Notre Dame, Indiana

April 2009

SOLUTION METHODS FOR NONLINEAR OPTIMAL CONTROL OF DISTRIBUTED ROBOTIC SYSTEMS

Abstract

by

Andrés K. Valenzuela

Three solution methods are considered for the optimal control problem of moving holonomic spherical robots from one configuration to another while maintaining the spacing between adjacent robots. First the optimal control problem is converted to a nonlinear boundary value problem through the calculus of variations. Then three numerical methods - the shooting method, a finite difference approximation solved through the Newton-Raphson method, and a finite difference approximation solved through homotopy continuation - are used to solve the boundary value problem. Both the shooting method and homotopy continuation found solutions to the optimal control problem. Homotopy continuation holds more promise for future research because of its ability to find all of the isolated solutions of the approximated system.

CHAPTER 1

Introduction

1.1 Motivation

Numerous control strategies for distributed robotic systems have garnered attention in the past several decades [2]. The applications of these strategies range from the control of autonomous underwater vehicles [5] to the movement of cars on the highway [12]. These systems, as well as many others, benefit from the robustness and flexibility of distributed systems.

As the number of agents in such systems grows, however, modeling their behavior, and simulating control techniques becomes computationally expensive. For systems comprising multiple identical agents (symmetric systems [9]), the characteristics of a system with only a few agents can in some cases reveal corresponding characteristics of a much larger system. McMickell and Goodwine [9] show that the controllability of symmetric systems can be determined in this manner. They go on to demonstrate the feasibility of using this reduction to perform motion planning for systems of identical nonholonomic robots [10].

This paper treats the investigation of another form of motion planning for a

system of identical holonomic robots. In this system, formation maintenance is achieved by finding the solution of the system's equations of motion that minimizes a weighted combination of the control effort and deviation from the desired formation. In looking for possible reductions due to the symmetry of the system, Deng, Sen, and Goodwine [3] found that as the weight given to maintaining formation increased, the system passed from having a unique optimal solution, to having multiple locally optimal solutions. This paper examines different methods for finding these multiple solutions.

1.2 Other work in the field

For a good, albeit slightly dated, survey of the field of distributed robotics as a whole, see [2]. In the area of path planning for formations of robots, four main approaches can be identified. In the first of these, the leader-follower approach [4], a motion plan for the formation as a whole is given by an external source. The lead robot follows this plan, and the other robots maintain the formation by following control laws based on their dynamics and their position in the formation. In the second method, based on reducing complexity of the system through symmetries [10] a path is planned for only one robot, and the paths for the remaining robots are worked out via the symmetric relationship between them. The virtual structure approach [8], a third method, is similar to the leader-follower, except that now all of the robots follow an imaginary rigid body in which they are "embedded." At each time step, the robots attempt to move into the new position of the virtual structure.

If they do not arrive, the structure readjusts so that it is as close as possible to being aligned with the robots.

The fourth approach is to use optimal control to generate the paths for the robots. The objective function used in the optimization is usually a scalar function, though vector objective functions have also been considered [7]. In this method, a system of differential equations representing the optimized system is solved to reveal the time-history of the state variables that leads to an optimal solution. This paper considers solution methods for the optimal control of a system of nonlinear robots through a scalar objective function.

CHAPTER 2

Description of Problem

2.1 The system

The distributed system considered here consists of N spherical robots on a plane. The control inputs for each robot are the x and y components of its velocity. Thus, for robot i,

$$u_{i,1} = \dot{x}_i \tag{2.1}$$

$$u_{i,2} = \dot{y}_i. \tag{2.2}$$

The robots begin in a configuration represented by the initial position vectors

$$\mathbf{x_a} = \begin{bmatrix} x_{a,1} \\ x_{a,2} \\ \vdots \\ x_{a,N} \end{bmatrix} \text{ and } \mathbf{y_a} = \begin{bmatrix} y_{a,1} \\ y_{a,2} \\ \vdots \\ y_{a,N} \end{bmatrix} \tag{2.3}$$

and end in a configuration represented by the terminal position vectors

$$\mathbf{x_b} = \begin{bmatrix} x_{b,1} \\ x_{b,2} \\ \vdots \\ x_{b,N} \end{bmatrix} \text{ and } \mathbf{y_b} = \begin{bmatrix} y_{b,1} \\ y_{b,2} \\ \vdots \\ y_{b,N} \end{bmatrix}. \tag{2.4}$$

The system seeks to achieve two objectives. The first is to minimize the total control effort necessary to move the robots to their terminal configuration. The second objective is to maintain a distance, $\bar{d}$, between each pair of adjacent robots. When no weight is given to this second objective, the robots follow straight-line trajectories, as shown in Figure 2.1 for the system considered in § 2.3. As the relative weight of the second objective is increased, the robots deviate from these trajectories. This work considers open chains of robots, that is, configurations in which the first and last robots are adjacent to only one other robot.

2.2 Derivation of differential equations

The objective function for the optimization of the robots' trajectory consists of two components that correspond to the two objectives described above. The component corresponding to the overall control effort is given by

$$J_1 = \int_0^1 \sum_{i=1}^{n} \left(\dot{x}_i^2 + \dot{y}_i^2\right) dt.$$

The component corresponding to the maintenance of the spacing between adjacent robots is given by

$$J_2 = \int_0^1 \sum_{i=1}^{N-1} \left(d_i - \bar{d}\right) dt,$$

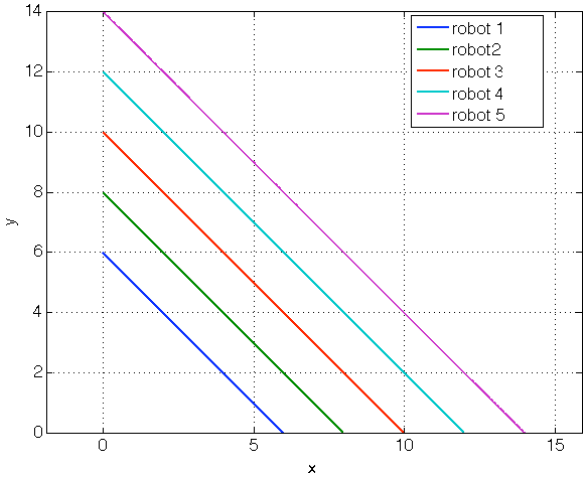

Figure 2.1. Trajectories of five spherical robots: Optimized for control effort only

where $d_i = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$. Thus the complete objective function is

$$\begin{aligned} J &= \int_0^1 \sum_{i=1}^{n} \left(\dot{x}_i^2 + \dot{y}_i^2\right) + \sum_{i=1}^{N-1} \left(d_i - \bar{d}\right) \\ &= \int_0^1 F(t, x_1, x_2, ..., x_N, y_1, y_2, ..., y_N, \dot{x}_1, \dot{x}_2, ..., \dot{x}_N, \dot{y}_1, \dot{y}_2, ..., \dot{y}_N,)dt, \end{aligned} \tag{2.5}$$

where k is a constant that defines the relative weight given to maintaining the spacing between adjacent robots.

We then use the calculus of variations to minimize J. A good description of this method can be found in [6]. It yields the following set of $2N$ differential equations:

$$\frac{\partial}{\partial x_i} F - \frac{d}{dt}\left(\frac{\partial}{\partial \dot{x}_i} F\right) = 0, \quad i = 1, 2, \ldots, n \tag{2.6}$$

and

$$\frac{\partial}{\partial y_i}F - \frac{d}{dt}\left(\frac{\partial}{\partial \dot{y}_i}F\right) = 0, \quad i = 1, 2, \ldots, n \tag{2.7}$$

Evaluating the derivatives in equations (2.6) and (2.7) leads to a system of $2N$ second order nonlinear ODEs. The second terms in equations (2.6) and (2.7) respectively become

$$\frac{d}{dt}\left(\frac{\partial}{\partial \dot{x}_i}F\right) = -2\ddot{x}_i, \; i = 1, 2, \ldots, n \tag{2.8}$$

and

$$\frac{d}{dt}\left(\frac{\partial}{\partial \dot{y}_i}F\right) = -2\ddot{y}_i, \; i = 1, 2, \ldots, n \tag{2.9}$$

respectively. For the interior robots, the first terms in equations (2.6) and (2.7) become

$$\frac{\partial}{\partial x_i}F = 2k\left[(x_i - x_{i-1})\frac{d_{i-1} - \bar{d}}{d_{i-1}} + (x_{i+1} - x_i)\frac{d_i - \bar{d}}{d_i}\right] \; i = 2, 3 \ldots, n-1 \tag{2.10}$$

and

$$\frac{\partial}{\partial y_i}F = 2k\left[(y_i - y_{i-1})\frac{d_{i-1} - \bar{d}}{d_{i-1}} + (y_{i+1} - y_i)\frac{d_i - \bar{d}}{d_i}\right] \; i = 2, 3 \ldots, n-1. \tag{2.11}$$

The first and last robot have only one neighbor each, and therefore, for these robots, the first terms in equations (2.6) and (2.7) evaluate to

$$\frac{\partial}{\partial x_1}F = 2k(x_2 - x_1)\frac{d_1 - \bar{d}}{d_1} \tag{2.12}$$

$$\frac{\partial}{\partial y_1}F = 2k(y_2 - y_1)\frac{d_1 - \bar{d}}{d_1} \tag{2.13}$$

$$\frac{\partial}{\partial x_N}F = 2k(x_N - x_{N-1})\frac{d_{N-1} - \bar{d}}{d_{N-1}} \tag{2.14}$$

$$\frac{\partial}{\partial y_N}F = 2k(y_N - y_{N-1})\frac{d_{N-1} - \bar{d}}{d_{N-1}}. \tag{2.15}$$

Substituting the expressions in equations (2.8) through (2.15) into (2.6) and (2.7) yields a system of equations that describe the optimal control problem. This system is given below in vector form.

$$
\begin{aligned}
\mathbf{g}(\mathbf{x}(t),\mathbf{y}(t)) &= \begin{bmatrix}
\ddot{x}_1 + k(x_2 - x_1)\frac{d_1-\bar{d}}{d_1} \\
\ddot{y}_1 + k(y_2 - y_1)\frac{d_1-\bar{d}}{d_1} \\
\ddot{x}_2 + k\left[(x_3 - x_2)\frac{d_2-\bar{d}}{d_2} - (x_2 - x_1)\frac{d_1-\bar{d}}{d_1}\right] \\
\ddot{y}_2 + k\left[(y_3 - y_2)\frac{d_2-\bar{d}}{d_2} - (y_2 - y_1)\frac{d_1-\bar{d}}{d_1}\right] \\
\vdots \\
\ddot{x}_i + k\left[(x_{i+1} - x_i)\frac{d_i-\bar{d}}{d_i} - (x_i - x_{i-1})\frac{d_{i-1}-\bar{d}}{d_{i-1}}\right] \\
\ddot{y}_i + k\left[(y_{i+1} - y_i)\frac{d_i-\bar{d}}{d_i} - (y_i - y_{i-1})\frac{d_{i-1}-\bar{d}}{d_{i-1}}\right] \\
\vdots \\
\ddot{x}_{N-1} + k\left[(x_N - x_{N-1})\frac{d_{N-1}-\bar{d}}{d_{N-1}} - (x_{N-1} - x_{N-2})\frac{d_{N-2}-\bar{d}}{d_{N-2}}\right] \\
\ddot{y}_{N-1} + k\left[(y_N - y_{N-1})\frac{d_{N-1}-\bar{d}}{d_{N-1}} - (y_{N-1} - y_{N-2})\frac{d_{N-2}-\bar{d}}{d_{N-2}}\right] \\
\ddot{x}_N - k(x_N - x_{N-1})\frac{d_{N-1}-\bar{d}}{d_{N-1}} \\
\ddot{y}_N - k(y_N - y_{N-1})\frac{d_{N-1}-\bar{d}}{d_{N-1}}
\end{bmatrix} \\
&= \mathbf{0}. \qquad (2.16)
\end{aligned}
$$

Together with equations (2.3) and (2.4), equation (2.16) gives a nonlinear boundary value problem (BVP) that represents the optimal control problem.

2.3 Specific System

The specific boundary conditions examined here are defined by the following initial and terminal positions.

$$\mathbf{x_a} = \mathbf{y_b} = \begin{bmatrix} 2\bar{d} \\ 2\bar{d} + \bar{d} \\ 2\bar{d} + 2\bar{d} \\ \vdots \\ 2\bar{d} + (i-1)\bar{d} \\ \vdots \\ 2\bar{d} + (N-1)\bar{d} \end{bmatrix} \text{ and } \quad \mathbf{x_b} = \mathbf{y_a} = \mathbf{0}. \tag{2.17}$$

At $t = 0$, the robots lie along the positive x-axis, each separated from the adjacent robots by a distance $\bar{d}$. The first robot starts a distance of $2\bar{d}$ from the origin. This initial configuration is shown in Figure 2.2.

The simulation runs until $t = 1$. At the end of the simulation, the robots are to reach a terminal configuration analogous to the initial configuration, but along the positive y-axis. This configuration is shown in Figure 2.2.

Thus, the nonlinear BVP defined by (2.16) and (2.17) represents the specific BVP considered here. The remaining chapters treat different methods of solving this BVP.

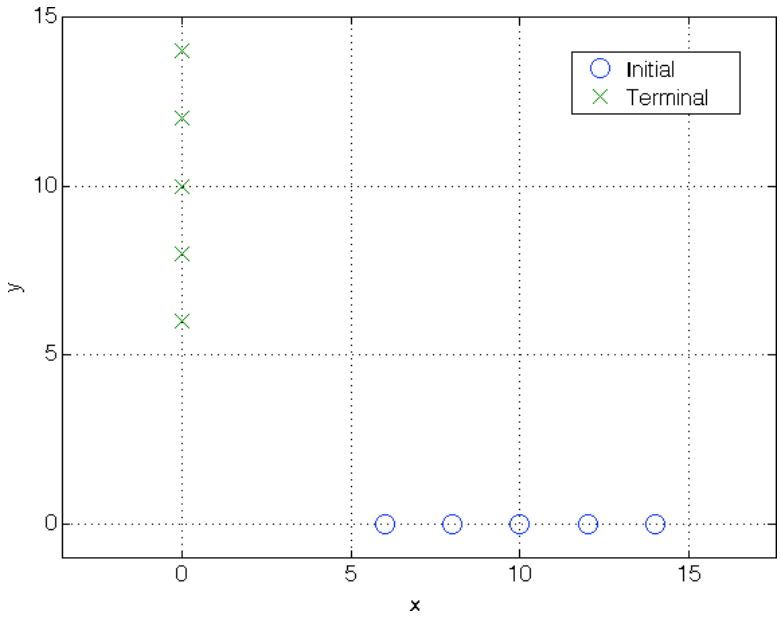

Figure 2.2. Initial and terminal configurations of the five spherical-robot system

CHAPTER 3

Shooting Method

3.1 Theoretical Background

In the shooting method, a BVP is converted to an initial value problem. Guesses at the unknown initial conditions supplement the known initial conditions in order to fully define the initial value problem. In the case of a BVP such as the one developed in the previous chapter, the known initial positions are supplemented with guesses for the unknown initial velocities. The initial value problem is then solved numerically to yield terminal values. These terminal values are compared to the desired terminal values and the guess at the initial velocities is adjusted. Through multiple iterations, a set of initial velocities is found which result in terminal values that match the desired values to a given level of accuracy.

Many different methods can be used to determine how the initial velocities should be adjusted so as to move the terminal positions closer to the desired endpoints. The method of adjoints is one such method. The description of the method of adjoints for nonlinear two point BVPs given below follows [11].

This method seeks to solve the following system of n nonlinear ODEs

$$\dot{q}_i = u_i(q_1, q_2, \ldots, q_n, t), \quad i = 1, 2, \ldots, n, \quad t_i \leq t \leq t_f \tag{3.1}$$

with the initial conditions

$$q_i(t_0) = c_i, \quad i = 1, 2, \ldots, r, \tag{3.2}$$

and the terminal conditions

$$q_{i_m}(t_f) = c_{i_m}, \quad m = 1, 2, \ldots, n - r. \tag{3.3}$$

Note that equations (3.1), (3.2), and (3.3) allow for initial and terminal values to be specified for any combination of variables. Some u_i may have only initial conditions specified, others may have only terminal conditions specified, while still others may have both initial and terminal conditions specified. All that is required is that the sum of the number of variables for which initial conditions are specified, r, and the number of variables for which terminal conditions are specified, m, be equal to the number of ODEs in the system, n.

Let $q_{i_{true}}(t), \quad i = 1, 2, \ldots, n$ be the solution to the boundary value problem. Consider a nearby solution to equation (3.1), $q_i(t), \quad i = 1, 2, \ldots, n$. The first-order correction (or variation), $\delta q_i(t)$, to the $q_i(t)$ is defined by

$$q_i(t) + \delta q_i(t) = q_{i_{true}}(t). \tag{3.4}$$

If a sufficiently good guess at the unknown initial conditions is used, the resulting solution can be taken to be $q_i(t)$ in equation (3.4). This method seeks to apply

incremental corrections to the guessed unknown initial conditions until the maximum value of the variation between known terminal values and the corresponding calculated terminal values is less than a certain tolerance.

The differential equation of the nearby solution is found by differentiating equation (3.4). This yields,

$$\dot{q}_i(t) + \delta\dot{q}_i(t) = u_i\left(q_1(t) + \delta q_1(t), q_2(t) + \delta q_2(t), \ldots, q_n(t) + \delta q_n(t), t\right), \quad i = 1, 2, \ldots, n \tag{3.5}$$

Expanding equation (3.5) in a Taylor series and discarding all but the first-order terms gives

$$\dot{q}_i(t) + \delta\dot{q}_i(t) = u_i\left(q_1(t), q_2(t), \ldots, q_n(t), t\right) + \sum_{j=1}^{n} \frac{\partial u_i}{\partial q_j}\delta q_j(t), \quad i = 1, 2, \ldots, n. \tag{3.6}$$

Subtracting equation (3.1) yields

$$\delta\dot{q}_i(t) = \sum_{j=1}^{n} \frac{\partial u_i}{\partial q_j}\delta q_j(t), \quad i = 1, 2, \ldots, n. \tag{3.7}$$

In vector form, this is

$$\delta\dot{\mathbf{q}} = J_{\mathbf{u}}^T \delta\mathbf{q}(t), \tag{3.8}$$

where $\mathbf{q}(t) = (q_1(t), q_2(t), \ldots, q_n(t))^T$, $\delta\mathbf{q}(t) = (\delta q_1(t), \delta q_2(t), \ldots, \delta q_n(t))^T$, and $\mathbf{u}(\mathbf{q}, t) = (u_1, u_2, \ldots, u_n)^T$.

The equations which are adjoint to the equations represented by (3.8) are given by

$$\dot{\mathbf{p}} = -J_{\mathbf{u}}\mathbf{p}. \tag{3.9}$$

Left multiplying equation (3.8) by $\mathbf{p}^T$ and equation (3.9) by $\delta\mathbf{q}^T$ yields

$$\mathbf{p}^T\delta\dot{\mathbf{q}} = \mathbf{p}^T J_{\mathbf{u}}^T \delta\mathbf{q}, \tag{3.10}$$

and

$$\delta\mathbf{q}^T\dot{\mathbf{p}} = -\delta\mathbf{q}^T J_{\mathbf{u}}\mathbf{p} \tag{3.11}$$

Taking the transpose of both sides of equation (3.11) and adding the result to equation (3.10) yields

$$\dot{\mathbf{p}}^T\delta\mathbf{q} + \mathbf{p}^T\delta\dot{\mathbf{q}} = \frac{d}{dt}\left(\mathbf{p}\delta\mathbf{q}\right) = 0. \tag{3.12}$$

Integrating with respect to time from t_i to t_f gives the fundamental identity of the method of adjoints

$$\mathbf{p}^T(t_f)\delta\mathbf{q}(t_f) - \mathbf{p}^T(t_0)\delta\mathbf{q}(t_0) = 0. \tag{3.13}$$

To relate the error in the terminal conditions to the required modification of the initial conditions, the adjoint equations (3.9) are numerically integrated from t_f to t_0, the reverse of the usual order. This integration is performed $n-r$ times with the Kronecker delta terminal conditions for $\mathbf{p}$:

$$p_i^{(m)}(t_f) = \left.\begin{array}{ll} 1, & i = i_m \\ 0, & i \neq i_m \end{array}\right\}, \quad m = 1, 2, \ldots, n-r, \tag{3.14}$$

where $p_i^{(m)}(t_f)$ is the terminal value of p_i for the m^{th} backwards integration and where $i_m,\ m = 1, 2, \ldots, n-r$ is the set of indices for which the terminal conditions

of the state variables are known. For the m^{th} set of terminal conditions and the l^{th} iteration then, equation (3.13) reduces to

$$\delta q_{i_m}^{(l)} - \sum_{i=1}^{n} p_i(t_0)\delta q_i^{(l)}(t_0) = 0, \;\; m = 1, 2, \ldots, n-r. \tag{3.15}$$

It should be noted that, for the variables whose initial conditions are known, $q_i(t_0),\ i = 1, 2, \ldots, r$

$$\delta q_i^{(l)}(t_0) = 0, \;\; i = 1, 2, \ldots, r, \tag{3.16}$$

since the variable will be set equal to the required initial condition at $t = t_0$. Therefore equation (3.15) becomes

$$\delta q_{i_m}^{(l)} - \sum_{i=r+1}^{n} p_i(t_0)\delta q_i^{(l)}(t_0) = 0, \;\; m = 1, 2, \ldots, n-r. \tag{3.17}$$

Equation (3.17) can be rewritten in matrix form as

$$\begin{bmatrix} p_{r+1}^{(1)}(t_0) & p_{r+2}^{(1)}(t_0) & \cdots & p_n^{(1)}(t_0) \\ p_{r+1}^{(2)}(t_0) & p_{r+2}^{(2)}(t_0) & \cdots & p_n^{(2)}(t_0) \\ \vdots & \vdots & & \vdots \\ p_{r+1}^{(n-r)}(t_0) & p_{r+2}^{(n-r)}(t_0) & \cdots & p_n^{(n-r)}(t_0) \end{bmatrix} \begin{bmatrix} \delta q_{r+1}^{(1)}(t_0) \\ \delta q_{r+2}^{(1)}(t_0) \\ \vdots \\ \delta q_n^{(1)}(t_0) \end{bmatrix} = \begin{bmatrix} \delta q_{i_1}^{(l)}(t_f) \\ \delta q_{i_2}^{(l)}(t_f) \\ \vdots \\ \delta q_{i_n-r}^{(l)}(t_f) \end{bmatrix}. \tag{3.18}$$

Solving yields the following equation which can be used to determine the required adjustment to the unknown initial conditions.

$$\begin{bmatrix} \delta q_{r+1}^{(1)}(t_0) \\ \delta q_{r+2}^{(1)}(t_0) \\ \vdots \\ \delta q_n^{(1)}(t_0) \end{bmatrix} = \begin{bmatrix} p_{r+1}^{(1)}(t_0) & p_{r+2}^{(1)}(t_0) & \cdots & p_n^{(1)}(t_0) \\ p_{r+1}^{(2)}(t_0) & p_{r+2}^{(2)}(t_0) & \cdots & p_n^{(2)}(t_0) \\ \vdots & \vdots & & \vdots \\ p_{r+1}^{(n-r)}(t_0) & p_{r+2}^{(n-r)}(t_0) & \cdots & p_n^{(n-r)}(t_0) \end{bmatrix}^{-1} \begin{bmatrix} \delta q_{i_1}^{(l)}(t_f) \\ \delta q_{i_2}^{(l)}(t_f) \\ \vdots \\ \delta q_{i_n-r}^{(l)}(t_f) \end{bmatrix}. \tag{3.19}$$

Therefore, the procedure for solving the BVP using the method of adjoints is as shown below.

1. Determine analitical expressions for $\frac{\partial u_i}{\partial q_j},\ i,j = 1,2,\ldots,n$.

2. Initialize the iteration counter: $l = 0$.

3. Guess the unknown initial conditions, $q_i^{(0)},\ i = r+1, r+2, \ldots, n$.

4. Integrate equation (3.1) from t_0 to t_f using known starting conditions and the guess from the previous step. Store the profiles.

5. Set the counter for the integration of the adjoint equations, $m = 1$

6. Integrate equation (3.9) from t_f to t_0 starting with the m^{th} set of terminal conditions in equation (3.14). Use the stored profiles, $q_i^{(l)},\ i = 1,2,\ldots,n$ and the analytical expressions for the partial derivates determined above to evaluate the Jacobian. Populate the m^{th} row of the matrix on the right hand side of equation (3.19) with $p_i^{(m)}(t_0),\ i = r+1, r+2, \ldots, n$.

7. To generate the m^{th} element of the vector on the right hand side of equation (3.19) take the difference between the desired terminal condition $q_{i_m}(t_f) = c_{i_m}$ and the calculated value $q_{i_,}^{(l)}(t_f)$ found in item 4.

8. If $m < n - r$, increment m and go to item 6.

9. Evaluate the right hand side of equation (3.19) to solve for $\delta q_i^{(l)}(t_0),\ i = r+1, r+2, \ldots, n$.

10. Generate a new guess for the unknown initial conditions according to

$$q_i^{(l+1)}(t_0) = q_i^{(l)}(t_0) + \delta q_i^{(l)}(t_0),\ i = r+1, r+2, \ldots, n. \tag{3.20}$$

11. If $\max\left\{\delta q_i^P(l)(t_f),\ m = 1,2,\ldots,n-r\right\} \geq \epsilon$, where ϵ is the desired tolerance for the solution, set $k = k+1$ and return to item 4. Otherwise, the l^{th} set of profiles is the numerical solution to the BVP.

3.2 Implementation

To represent the N-robot system succinctly, the x and y components of the robots' positions and velocities are concatenated to create a new variable, given by

$$z_i = \begin{cases} x_i, & i = 1, 2, \dots N \\ y_{i-N}, & i = n+1, n+2, \dots, 2N \\ \dot{x}_{i-2N}, & i = 2N+1, 2N+2, \dots, 3N \\ \dot{y}_{i-3N}, & i = 3N+1, 3N+2, \dots, 4N. \end{cases}$$

The system of ODEs is therefore

$$\dot{\mathbf{z}} = \mathbf{g}(\mathbf{z}, t), \tag{3.21}$$

where $\mathbf{g}(\mathbf{z}, t) = (g_1, g_2, \dots, g_{4N})^T$ is given by

$$g_i(\mathbf{z}, t) = \begin{cases} z_{i+2N}, & i = 1, 2, \dots, 2N \\ -k(z_2 - z_1)\frac{d_1 - \bar{d}}{d_1}, & i = 2N+1 \\ k\left[(z_{i-2N} - z_{i-2N-1})\frac{d_{i-2N-1} - \bar{d}}{d_{i-2N-1}} - (z_{i-2N+1} - z_{i-2N})\frac{d_{i-2N} - \bar{d}}{d_{i-2N}}\right], & i = 2N+2, \dots, 3N-1 \\ k(z_N - z_{N-1})\frac{d_{N-1} - \bar{d}}{d_{N-1}}, & i = 3N \\ -k(z_{N+2} - z_{N+1})\frac{d_1 - \bar{d}}{d_1}, & i = 3N+1 \\ k\left[(z_{i-2N} - z_{i-2N-1})\frac{d_{i-3N-1} - \bar{d}}{d_{i-3N-1}} - (z_{i-2N+1} - z_{i-2N})\frac{d_{i-3N} - \bar{d}}{d_{i-3N}}\right], & i = 3N+2, \dots, 4N-1 \\ k(z_{2N} - z_{2N-1})\frac{d_{N-1} - \bar{d}}{d_{N-1}}, & i = 4N \end{cases} \tag{3.22}$$

The initial and terminal conditions are known for z_1 through z_{2N} and are given by

$$z_{a,i} = \begin{cases} x_{a,i}, & i = 1, 2, \dots, N \\ y_{a,i-N}, & i = N+1, N+2, \dots, 2N \end{cases} \tag{3.23}$$

and

$$z_{b,i} = \begin{cases} x_{b,i}, & i = 1, 2, \ldots, N \\ y_{b,i-N}, & i = N+1, N+2, \ldots, 2N, \end{cases} \tag{3.24}$$

respectively. Substituting $\mathbf{g}(\mathbf{z}, t)$ for $\mathbf{u}(\mathbf{q}, t)$ and setting $n = 4N$, $r = 2N$ allows us to solve the BVP given in (3.1), (3.23), and (**??**) using the method described in § 3.1.

In order implement that method, however, we must first find an analytical expression for the Jacobian of $\mathbf{g}(\mathbf{z}, t)$. Evaluating the partial derivatives yields the expression for $J_{\mathbf{g}}(\mathbf{z}, t)$ found in equations (3.25) through (3.32).

$$J_{\mathbf{g}} = \begin{bmatrix} A & B & C \end{bmatrix} \tag{3.25}$$

where, for $i = 1, 2, \ldots, 4N$,

$$a_{i,j} = \begin{cases} 1, & i = j + 2N \\ 0, & i \neq j + 2N \end{cases} \quad j = 1, 2, \ldots, 2N, \tag{3.26}$$

$$
b_{i,j} = \begin{cases}
\left.\begin{array}{rl} 0, & i \neq 1,2,N,N+1 \\ -kf_1(\mathbf{z},i), & i=1 \\ kf_1(\mathbf{z},i-1), & i=2 \\ -kf_2(\mathbf{z},i), & i=N \\ kf_2(\mathbf{z},i-1), & i=N+1 \end{array}\right\} j=1 \\
\left.\begin{array}{rl} 0, & i \neq \begin{array}{r} j, j\pm 1, \\ N+j, \\ N+j\pm 1 \end{array} \\ kf_1(\mathbf{z},i), & i=j-1 \\ -k\left[f_1(\mathbf{z},i-1)+f_1(\mathbf{z},i)\right], & i=j \\ kf_1(\mathbf{z},i-1) & i=j+1 \\ kf_2(\mathbf{z},i), & i=N+j-1 \\ -k\left[f_2(\mathbf{z},i-1)+f_2(\mathbf{z},i)\right], & i=N+j \\ kf_2(\mathbf{z},i-1) & i=N+j+1 \end{array}\right\} j=2,3,\ldots,N-1, \\
\left.\begin{array}{rl} 0, & i \neq j-1, j, N+j-1, N+j \\ kf_1(\mathbf{z},i), & i=j-1 \\ -kf_1(\mathbf{z},i-1), & i=j \\ kf_2(\mathbf{z},i), & i=N+j-1 \\ -kf_2(\mathbf{z},i-1), & i=N+j \end{array}\right\} j=N
\end{cases} \tag{3.27}
$$

$$
c_{i,j} \;=\; \begin{cases}
\left.\begin{array}{rl} 0, & i \neq 1,2,N,N+1 \\ -kf_3(\mathbf{z},i), & i=1 \\ kf_3(\mathbf{z},i-1), & i=2 \\ -kf_4(\mathbf{z},i), & i=N \\ kf_4(\mathbf{z},i-1), & i=N+1 \end{array}\right\} j=1 \\[2ex]
\left.\begin{array}{rl} & j, j\pm 1, \\ 0, & i \neq N+j, \\ & N+j\pm 1 \\ kf_3(\mathbf{z},i), & i=j-1 \\ -k\left[f_3(\mathbf{z},i-1)+f_3(\mathbf{z},i)\right], & i=j \\ kf_3(\mathbf{z},i-1) & i=j+1 \\ kf_4(\mathbf{z},i), & i=N+j-1 \\ -k\left[f_4(\mathbf{z},i-1)+f_4(\mathbf{z},i)\right], & i=N+j \\ kf_4(\mathbf{z},i-1) & i=N+j+1 \end{array}\right\} j=2,3,\ldots,N-1, \\[2ex]
\left.\begin{array}{rl} 0, & i \neq j-1, j, N+j-1, N+j \\ kf_3(\mathbf{z},i), & i=j-1 \\ -kf_3(\mathbf{z},i-1), & i=j \\ kf_4(\mathbf{z},i), & i=N+j-1 \\ -kf_4(\mathbf{z},i-1), & i=N+j \end{array}\right\} j=N
\end{cases} \tag{3.28}
$$

$$f_1(\mathbf{z},\alpha) = (z_{\alpha+1}-z_\alpha)^2\left[\frac{d_\alpha-\bar{d}}{d_\alpha^3}-\frac{1}{d_\alpha^2}\right]-\frac{d_\alpha-\bar{d}}{d_\alpha}, \tag{3.29}$$

$$f_2(\mathbf{z},\alpha) = (z_{\alpha+1}-z_\alpha)(z_{N+\alpha+1}-z_{N+\alpha})\left[\frac{d_\alpha-\bar{d}}{d_\alpha^3}-\frac{1}{d_\alpha^2}\right], \tag{3.30}$$

$$f_3(\mathbf{z},\alpha) = (z_{N+\alpha+1}-z_{N+\alpha})^2\left[\frac{d_\alpha-\bar{d}}{d_\alpha^3}-\frac{1}{d_\alpha^2}\right]-\frac{d_\alpha-\bar{d}}{d_\alpha}, \tag{3.31}$$

and

$$f_4(\mathbf{z},\alpha) = f_2(\mathbf{z},\alpha,\alpha+1). \tag{3.32}$$

Note that A is $4N\times 2N$, while B and C are both $4N\times N$. Thus, $J_{\mathbf{g}}$ is $4N\times 4N$ as it should be.

A MATLAB code (See A.1) implements the method described in § 3.1. The core of the code is a general shooting method solver. The solver takes a function representing $\mathbf{u}(\mathbf{q},t)$ from equation (3.1), a function that returns the Jacobian of u, vectors containing initial conditions and terminal conditions, and vectors indicating for which variables initial and terminal conditions are supplied, along with other parameters such as the time range and tolerance, as inputs. It returns a vector of time steps between t_0 and t_f, an array containing the calculated values of the state variables at each time step, and a vector containing the guess at the unknown initial conditions that generated the solution.

For the solution of the N-robot system, the MATLAB functions that represent $\mathbf{u}(\mathbf{q},t)$ and its Jacobian implement equations (3.22) and (3.25) respectively. The

known initial and terminal conditions are taken from (2.17). However, because the analysis in § 3.1 is based on the assumption that $\mathbf{q}$ lies near $\delta\mathbf{q}_{true}$, the method may not lead to a solution if the first guess at initial conditions leads is too far off. The code overcomes this difficulty by dividing the distance between the terminal conditions yielded by the first iteration and the desired terminal condition into a number of equal steps. The shooting method is then implemented repeatedly, each repetition's desired terminal conditions moving one step closer to the terminal conditions desired overall. The final value of the unknown initial conditions for one iteration serves as an initial guess for the next. This makes it more likely that the initial solution to the ODE will be close enough to the solution of the BVP for it to converge to that solution.

3.3 Results

In order to compare solutions to the BVP, we use the following metric taken from [3]. The straight line solution between the initial and terminal position of each robot provides a base line against which to measure different paths for that robot. Each path is characterized by its deviation, d_{dev}, from the straight line solution at a time $t^\star$. This provides some indication of the difference between paths, as long as the two paths do not cross at that time. The sign of d_{dev} is determined by whether or not the position of the robot at time $t^\star$ lies above or below the straight line path. For the system N-robot system described in § 2.3, this yields the following definition of

the deviation of the i^{th} robot:

$$d_{devi}(t^\star) = \begin{cases} -\sqrt{(x_i(t^\star) - x_{nom,i}(t^\star))^2 + y_i(t^\star) - y_{nom,i}(t^\star))^2}, \\ \qquad \text{if } y(t^\star) < y_{a,i} + (x_i(t^\star) - x_{a,i})\,\frac{y_{b,i}-y_{a,i}}{x_{b,i}-x_{a,i}} \\ \sqrt{(x_i(t^\star) - x_{nom,i}(t^\star))^2 + y_i(t^\star) - y_{nom,i}(t^\star))^2}, \\ \qquad \text{if } y(t^\star) < y_{a,i} + (x_i(t^\star) - x_{a,i})\,\frac{y_{b,i}-y_{a,i}}{x_{b,i}-x_{a,i}}, \end{cases} \tag{3.33}$$

where x_i and y_i are the coordinates of the robot on a given path and $x_{nom,i}$ and $y_{nom,i}$ are the coordinates of a robot on the nominal straight line path.

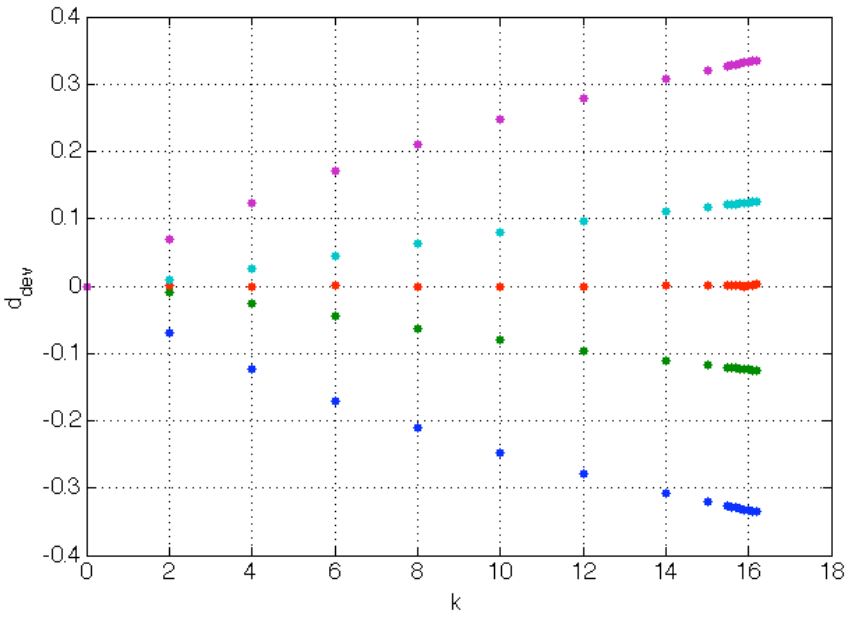

Figure 3.1. Plot of deviation from the straight-line path at $t = 0.25$ vs. k for $k < 16$

Figure 3.1 shows the metric d_{dev} for low k values. For k between 0 and approx-

imately 16.5, $d_{devi}(0.25)$ follows a single continuous path for each robot. Between $k = 16$ and $k \approx 16.5$, however, $d_{devi}(0.25)$ jumps to a new value for all of the robots, as shown in Figure 3.2.

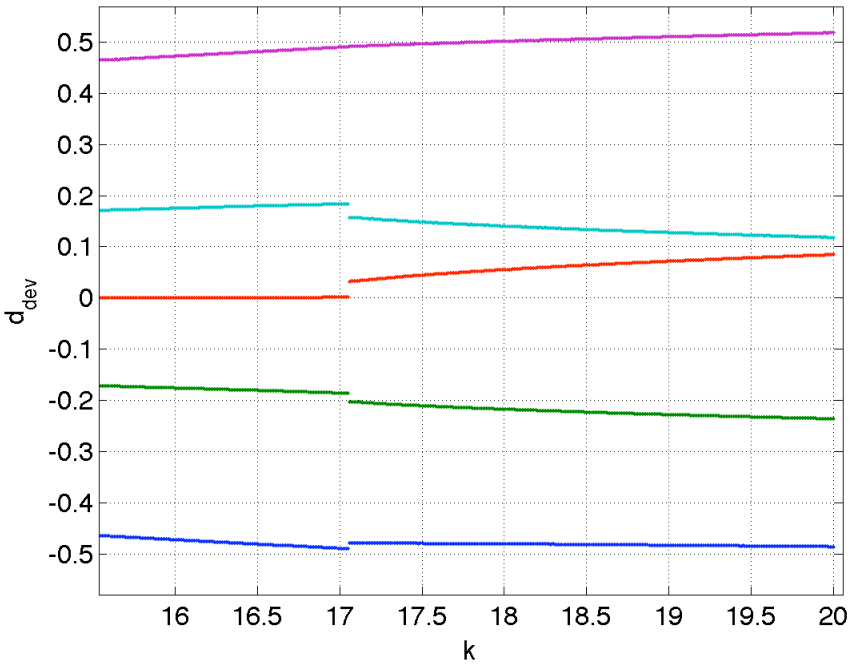

Figure 3.2. Plot of deviation from the straight-line path at $t = 0.25$ vs. k

A likely cause for this is the shooting method solver jumping from one solution to another at that point. However, we were unable to find the continuation of the original solution past that point, so this jump is not a conclusive confirmation of Deng's numerical results [**?**]. Note that the d_{dev} plotted in Figure 3.2 is a slightly different metric than the d_{dev} plotted in Figure 3.1. The implementation of the

shooting method was the same for both simulations, however, so the qualitative change in the plots of d_{dev} shown in Figure 3.2 does represent a change in the solutions to the system assessed in Figure 3.1.

The shooting method fails at high k values (for a five robot system: above $k \approx 20$). At these values the process does not converge towards a solution that satisfies the terminal conditions, but rather diverges.

CHAPTER 4

Finite Difference Method

4.1 Theoretical Background

In the finite difference method employed here, a two-point BVP

$$\begin{aligned}
\mathbf{f}\left(t, \mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t)\right) &= \begin{bmatrix} f_1\left(t, \mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t)\right) \\ f_2\left(t, \mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t)\right) \\ \vdots \\ f_i\left(t, \mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t)\right) \\ \vdots \\ f_p\left(t, \mathbf{q}(t), \dot{\mathbf{q}}(t), \ddot{\mathbf{q}}(t)\right) \end{bmatrix} = \mathbf{0},\ t \in [t_a, t_b] \qquad (4.1) \\
\mathbf{q}(t_a) &= \mathbf{q}_a \\
\mathbf{q}(t_b) &= \mathbf{q}_b
\end{aligned}$$

is converted to a system of algebraic equations by discretizing time into a series of equidistant mesh points $t_a = t_0, t_1, ..., t_m = t_b$, where $[t_a, t_b]$ is the interval over which the differential equation is to be solved. For the i^{th} component of $\mathbf{q}$, the first

derivative at time t_j is given by

$$\dot{q}_i(t_j) = \frac{q_i^{(j+1)} - q_i^{(j-1)}}{2h} + \mathcal{O}(h^2), \tag{4.2}$$

where $q_i^{(j)} = q_i(t_j)$ and h is the time between mesh points. The second derivative at the same time is given by

$$\ddot{q}_i(t^j) = \frac{q_i^{(j-1)} - 2q_i^{(j)} + q_i^{(j+1)}}{h^2} + \mathcal{O}(h^2). \tag{4.3}$$

The first terms of the right hand sides of equations (4.2) and (4.3) can be used to generate approximate equations for each component of $\mathbf{f}$ at each interior mesh point. If $\hat{\mathbf{q}}$ is defined by

$$\hat{\mathbf{q}} = \left[q_1^{(0)}, q_1^{(1)}, \ldots, q_1^{(m)}, q_2^{(0)}, \ldots, q_2^{(m)}, \ldots, q_p^{(0)}, \ldots, q_p^{(m)}\right]^T, \tag{4.4}$$

these approximate equations can be expressed in vector form as

$$\hat{\mathbf{f}}(\hat{\mathbf{q}}) = \begin{bmatrix} \hat{f}_{1,0}\left(\hat{\mathbf{q}}(t)\right) \\ \hat{f}_{1,1}\left(\hat{\mathbf{q}}(t)\right) \\ \vdots \\ \hat{f}_{1,m}\left(\hat{\mathbf{q}}(t)\right) \\ \hat{f}_{2,0}\left(\hat{\mathbf{q}}(t)\right) \\ \vdots \\ \hat{f}_{2,m}\left(\hat{\mathbf{q}}(t)\right) \\ \vdots \\ \hat{f}_{p,0}\left(\hat{\mathbf{q}}(t)\right) \\ \vdots \\ \hat{f}_{p,m}\left(\hat{\mathbf{q}}(t)\right) \end{bmatrix} = \mathbf{0} \tag{4.5}$$

where

$$\hat{f}_{i,0} = q_{a,i} - q_i^{(0)}, \tag{4.6}$$

$$\hat{f}_{i,m} = q_{b,i} - q_i^{(m)}, \tag{4.7}$$

and

$$\hat{f}_{i,j} = f_i \left(t_j, \begin{bmatrix} q_1^{(j)} \\ q_2^{(j)} \\ \vdots \\ q_p^{(j)} \end{bmatrix}, \begin{bmatrix} \frac{q_1^{(j+1)} - q_1^{(j-1)}}{2h} \\ \frac{q_2^{(j+1)} - q_2^{(j-1)}}{2h} \\ \vdots \\ \frac{q_p^{(j+1)} - q_p^{(j-1)}}{2h} \end{bmatrix}, \begin{bmatrix} \frac{q_1^{(j-1)} - 2q_1^{(j)} + q_1^{(j+1)}}{h^2} \\ \frac{q_2^{(j-1)} - 2q_2^{(j)} + q_2^{(j+1)}}{h^2} \\ \vdots \\ \frac{q_p^{(j-1)} - 2q_p^{(j)} + q_p^{(j+1)}}{h^2} \end{bmatrix} \right), \tag{4.8}$$

for $i = 1, \ldots, p$ and $j = 1, \ldots, m-1$. Equation (4.5) can now be solved using any number of methods for solving systems of nonlinear equations.

The method used here to solve equation (4.5) is a simple Newton's Method. It begins with a guess at the values of the positions at all mesh points, $\hat{\mathbf{q}}^{\mathbf{0}}$. A new value for $\hat{\mathbf{q}}$ is found through the following expression

$$\hat{\mathbf{q}}^{\mathbf{k+1}} = \mathbf{q}^{\mathbf{k}} - \lambda J_{\hat{f}}^{-1}(\hat{\mathbf{q}}^{\mathbf{k}}) \hat{\mathbf{f}}(\hat{\mathbf{q}}^{\mathbf{k}}), \tag{4.9}$$

where $\lambda \in (0, 1]$. Equation (4.9) is evaluated repeatedly until $\left\| \hat{\mathbf{q}}^{\mathbf{k+1}} - \hat{\mathbf{q}}^{\mathbf{k}} \right\| < \epsilon$, where ϵ is a specified convergence tolerance. $\hat{\mathbf{q}}^{\mathbf{k+1}}$ is the approximate solution for the system.

4.2 Implementation

The BVP represented by equations (2.16) and (2.17) has the appropriate form for the method described in § 4.1 if $\mathbf{g}(\mathbf{x}, \mathbf{y})$ is replaced with $\mathbf{g}(\mathbf{z})$, where $\mathbf{z}$ is now

given by

$$\mathbf{z}(t) = \begin{bmatrix} \mathbf{x}(t) \\ \mathbf{y}(t) \end{bmatrix}. \tag{4.10}$$

This yields the following BVP

$$\begin{aligned} \mathbf{g}(\mathbf{z}(\mathbf{t})) &= \mathbf{0}, t \in [t_a, t_b] \\ \mathbf{z_a} &= \begin{bmatrix} \mathbf{x_a} \\ \mathbf{y_a} \end{bmatrix}, \text{ and } \mathbf{z_b} = \begin{bmatrix} \mathbf{x_b} \\ \mathbf{y_b} \end{bmatrix} \end{aligned} \tag{4.11}$$

The finite difference approximation for this BVP is given by

$$\hat{g}_{ij}(\hat{\mathbf{z}}) = \begin{cases} z_{a,i} - \hat{z}_i^{(j)}, & j = 0 \\ \left.\begin{array}{ll} w_1(\hat{\mathbf{z}}, i, j), & i = 1, N+1 \\ w_2(\hat{\mathbf{z}}, i, j), & i = 2, 3, \ldots, N-1, N+2, N+3, \ldots, 2N-1 \\ w_3(\hat{\mathbf{z}}, i, j), & i = N, 2N \end{array}\right\} & j = 1, 2, \ldots, m-1 \\ z_{b,i} - \hat{z}_i^{(j)}, & j = m, \end{cases} \tag{4.12}$$

for $i = 1, 2, \ldots, 2N$ and $j = 0, 1, 2, \ldots, m$, where,

$$\begin{aligned} w_1(\hat{\mathbf{z}}, i, j) &= \frac{\hat{z}_i^{(j+1)} - 2\hat{z}_i^{(j)} + \hat{z}_i^{(j-1)}}{h^2} + k\left(\hat{z}_{i+1}^{(j)} - \hat{z}_i^{(j)}\right)\frac{d_i^{(j)} - \bar{d}}{d_i^{(j)}}, \\ w_2(\hat{\mathbf{z}}, i, j) &= \frac{\hat{z}_i^{(j+1)} - 2\hat{z}_i^{(j)} + \hat{z}_i^{(j-1)}}{h^2} + k\left[\left(\hat{z}_{i+1}^{(j)} - \hat{z}_i^{(j)}\right)\frac{d_i^{(j)} - \bar{d}}{d_i^{(j)}}\right. \\ &\qquad \left. - \left(\hat{z}_i^{(j)} - \hat{z}_{i-1}^{(j)}\right)\frac{d_{i-1}^{(j)} - \bar{d}}{d_{i-1}^{(j)}}\right], \\ w_3(\hat{\mathbf{z}}, i, j) &= \frac{\hat{z}_i^{(j+1)} - 2\hat{z}_i^{(j)} + \hat{z}_i^{(j-1)}}{h^2} - k\left(\hat{z}_i^{(j)} - \hat{z}_{i-1}^{(j)}\right)\frac{d_{i-1}^{(j)} - \bar{d}}{d_{i-1}^{(j)}}, \end{aligned}$$

and

$$d_i^{(j)} = \begin{cases} \sqrt{\left(z_{i+1}^{(j)} - z_i^{(j)}\right)^2 + \left(z_{N+i+1}^{(j)} - z_{N+i}^{(j)}\right)^2}, & i < N \\ \sqrt{\left(z_{i+1-N}^{(j)} - z_{i-N}^{(j)}\right)^2 + \left(z_{i+1}^{(j)} - z_i^{(j)}\right)^2}, & i > N \end{cases},$$

for $i \in \{1, 2, \ldots, N-1, N+1, N+2, \ldots, 2N-1\}$ and $j \in \{0, 1, \ldots, m\}$. If $\hat{\mathbf{g}}$ and $\hat{\mathbf{z}}$ are substituted for $\hat{\mathbf{f}}$ and $\hat{\mathbf{q}}$ in equation (4.9), that equation can be iterated from an initial guess at the trajectories of each robot, $\hat{\mathbf{z}}^0$ to yield a solution for (4.11). To increase the speed of computation, the Jacobian of $\hat{\mathbf{g}}$ can be computed analytically.

4.3 Results

We were unable to find a constant value of λ that provided led to convergence of $\hat{\mathbf{z}}$ for k values high enough to observe multiple solutions. When attempting to solve with the straight line solution as the initial guess, for any given λ there is a $k^\star$ such that when $k > k^\star$ the FDM code diverges. When stepping through k values beginning from $k = 0$ and using the solution at one k value as the initial guess for the next, the FDM code at some point enters a loop in which it cycles through the same set of $\hat{\mathbf{z}}$ values repeatedly. These two results make the use of the simple Newton's method to solve the finite difference approximation of the system infeasible. §5 describes a superior method for solving the finite difference approximation.

CHAPTER 5

Homotopy Method

5.1 Background

The greatest disadvantage of the shooting method and finite difference methods is their local nature. Since their search schemes involve changing parameters based on the current state of the system, such methods find only solutions that are close to the initial guess given them by the user. For the N-robot system, being able to be certain that one has found all of the solutions for a given k value would be beneficial in investigating the transition from a unique solution to multiple solutions. Numerical solvers that utilize homotopy continuation, such as Bertini [13], can find all of the isolated solutions of systems of polynomials. If the finite difference approximation of the N-robot optimal control problem can be converted into a polynomial system, then we should be able to find all physically relevant solutions for any given value of k.

The theory of homotopy continuation falls outside of the scope of this paper, but an explanation of it can be found in [14]. At a basic level, homotopy continuation for a given polynomial system takes a system of the same form as the one to be solved,

but with known roots, and transforms it continuously into the polynomial system with unknown roots. As it does this, it tracks the roots of the polynomial system, which ultimately become the roots of the given polynomial system.

Bates, Fotiou, and Rotalski [1] present a method for implementing nonlinear constrained optimal control using homotopy continuation. They consider the minimization of a polynomial objective function of a state-variable vector, z, and a parameter vector, x_0, that is subject to polynomial equality and inequality constraints. This leads to a system of polynomial equations, $\mathcal{F}_{x_0}$. In order to ease the computational burden associated with finding a solution to the optimization problem, $\mathcal{F}_{\hat{x}_0}$, for a given set of parameters, $\hat{x}_0$ they divide the computation in to two segments. The first,"off-line" part, need only be done once. In the off-line part, a solution $\mathcal{F}_{\tilde{x}_0}$ is found for a random parameter vector, $\tilde{x}_0$. This solution is found through homotopy continuation from a system that only resembles $\mathcal{F}$ at a basic level. Because of this, tracking the roots to find $\mathcal{F}_{\tilde{x}_0}$ is very computationally expensive. The second, "on-line" part finds $\mathcal{F}_{\hat{x}_0}$ through homotopy continuation from $\mathcal{F}_{\tilde{x}_0}$. Because $\mathcal{F}_{\tilde{x}_0}$ and $\mathcal{F}_{\hat{x}_0}$ result from the same naturally parameterized family, this homotopy continuation is much faster than that of the off-line part. Thus, once $\mathcal{F}_{\tilde{x}_0}$ has been computed, $\mathcal{F}_{\hat{x}_0}$ can be found relatively quickly.

Unfortunately, this approach to the use of homotopy methods was found to late to be employed in the research presented here. The subsequent sections detail the implementation and results of a homotopy continuation method that resembles the off-line part of the method presented above.

5.2 Implementation

As in §4, we discretize the interval $[t_a, t_b]$ into an equidistant mesh, $t_a = t_0, t_1, \ldots, t_m = t_b$. Once again, we approximate the second derivative terms of (2.16) by first term of equation (4.3). This yields a system of $2N(m-1)$ algebraic equations in the variables $x_1^{(1)}, y_1^{(1)}, x_1^{(2)}, y_1^{(2)}, \ldots, x_1^{(m-1)}, y_1^{(m-1)}, x_2^{(1)}, y_2^{(1)}, \ldots, x_2^{(m-1)}, y_2^{(m-1)}, \ldots, x_N^{(1)}, y_N^{(1)}, \ldots, x_N^{(m-1)}, y_N^{(m-1)}$. These equations are not polynomial equations, however. Therefore, we must convert that system into a system of polynomial equations before we can use Bertini to solve them.

This is accomplished by making $d_i^{(j)}$ a variable rather than a function of $x_i^{(j)}$ and $y_i^{(j)}$. The relationship between $d_i^{(j)}$,$x_i^{(j)}$, and $y_i^{(j)}$ is maintained by adding an equation to the system for each $d_i^{(j)}$. Making this change and clearing denominators yields,

$$
\begin{array}{lll}
\widetilde{f}_{1,1} = 0 & \vdots & \widetilde{h}_{1,1} = 0 \\
\widetilde{g}_{1,1} = 0 & \widetilde{f}_{2,m-1} = 0 & \widetilde{h}_{1,2} = 0 \\
\widetilde{f}_{1,2} = 0 & \widetilde{g}_{2,m-1} = 0 & \vdots \\
\widetilde{g}_{1,2} = 0 & \vdots & \widetilde{h}_{1,m-1} = 0 \\
\vdots & \vdots & \widetilde{h}_{2,1} = 0 \\
\widetilde{f}_{1,m-1} = 0 & \widetilde{f}_{N,1} = 0 & \vdots \\
\widetilde{g}_{1,m-1} = 0 & \widetilde{g}_{N,1} = 0 & \widetilde{h}_{2,m-1} = 0 \\
\widetilde{f}_{2,1} = 0 & \vdots & \vdots \\
\widetilde{g}_{2,1} = 0 & \widetilde{f}_{N,m-1} = 0 & \widetilde{h}_{N-1,1} = 0 \\
\vdots & \widetilde{g}_{N,m-1} = 0 & \vdots \\
 & & \widetilde{h}_{N-1,m-1} = 0,
\end{array}
$$

where

$$\widetilde{f}_{i,j} = \begin{cases} \left(x_i^{(j+1)} - 2x_i^{(j)} + x_i^{(j-1)}\right) d_i^{(j)} + h^2 k \left(x_{i+1}^{(j)} - x_i^{(j)}\right) \left(d_i^{(j)} - \bar{d}\right), & i = 1 \\ \left(x_i^{(j+1)} - 2x_i^{(j)} + x_i^{(j-1)}\right) d_i^{(j)} d_{i-1}^{(j)} & \\ \quad + h^2 k \left[\left(x_{i+1}^{(j)} - x_i^{(j)}\right) \left(d_i^{(j)} - \bar{d}\right) d_{i-1}^{(j)}\right. & i = 2, \ldots, N-1 \\ \quad \left. - \left(x_i^{(j)} - x_{i-1}^{(j)}\right) \left(d_{i-1}^{(j)} - \bar{d}\right) d_i^{(j)}\right], & \\ \left(x_i^{(j+1)} - 2x_i^{(j)} + x_i^{(j-1)}\right) d_{i-1}^{(j)} + h^2 k \left(x_i^{(j)} - x_{i-1}^{(j)}\right) \left(d_{i-1}^{(j)} - \bar{d}\right), & i = N, \end{cases} \quad (5.1)$$

$$\widetilde{g}_{i,j} = \begin{cases} \left(y_i^{(j+1)} - 2y_i^{(j)} + y_i^{(j-1)}\right) d_i^{(j)} + h^2 k \left(y_{i+1}^{(j)} - y_i^{(j)}\right) \left(d_i^{(j)} - \bar{d}\right), & i = 1 \\ \left(y_i^{(j+1)} - 2y_i^{(j)} + y_i^{(j-1)}\right) d_i^{(j)} d_{i-1}^{(j)} & \\ \quad + h^2 k \left[\left(y_{i+1}^{(j)} - y_i^{(j)}\right) \left(d_i^{(j)} - \bar{d}\right) d_{i-1}^{(j)}\right. & i = 2, \ldots, N-1 \\ \quad \left. - \left(y_i^{(j)} - y_{i-1}^{(j)}\right) \left(d_{i-1}^{(j)} - \bar{d}\right) d_i^{(j)}\right], & \\ \left(y_i^{(j+1)} - 2y_i^{(j)} + y_i^{(j-1)}\right) d_{i-1}^{(j)} + h^2 k \left(y_i^{(j)} - y_{i-1}^{(j)}\right) \left(d_{i-1}^{(j)} - \bar{d}\right), & i = N, \end{cases} \quad (5.2)$$

and

$$\widetilde{h}_{i,j} = {d_i^{(j)}}^2 - \left(x_{i+1}^{(j)} - x_i^{(j)}\right)^2 \left(y_{i+1}^{(j)} - y_i^{(j)}\right)^2. \quad (5.3)$$

A Perl script generates an input file containing this system in a format that Bertini can read.

5.3 Results

For a five-robot system with $k = 5$, Bertini produces a single finite-real solution. However, we have only found this solution over a four point time discretization. Attempts to solve the system over a more densely populated mesh require an unreasonable amount of time to run on a single computer. We have not yet sought to solve these denser mesh systems on a cluster due to conflicts between the available distributions of Bertini and the hardware of easily accessible clusters.

CHAPTER 6

Future Directions and Conclusions

6.1 Directions for future research

We were unable to run Bertini for systems with more than four mesh points for this paper due to time constraints and limitations on readily available computing resources. The next step is to solve the BVP with an algebro-geometric solver such as Bertini over meshes of useful density. This will require the use of parallel computing, as the total number of solutions (not limited to physically meaningful ones) grows very quickly for the N-robot system. Additionally, we should adopt a two part procedure along the lines of the one described in [1] and summarized in §5.1. This could greatly reduce the amount of computational effort required to find solutions for the system.

The comparison between distinct paths of a given robot is another area that future work on this topic should examine in greater depth. The d_{dev} metric used in this paper is helpful in displaying symmetry between the solutions, but has several significant drawbacks. As noted in § 3.3, it cannot differentiate between paths that cross at $t^\star$. Moreover, it does not give any information about the rest of the solution.

While loss of data is inevitable in moving from a multipoint two-dimensional path to a scalar metric, we should seek a metric that incorporates more of the data, so as to be better able to compare paths which are close to one another when $t = t^\star$. Perhaps an ordered pair of integrals, the first representing the area between the path and the straight-line path to one side of the straight-line path and the second representing the area between them to the other side of the straight line path. This and other possible metrics should be investigated.

6.2 Conclusions

Three approaches to the solution of the an optimal control problem for a particular distributed robotic system have been presented. While the results generated by the shooting method currently cover the most extensive range of parameter values, the shooting method has substantial unaddressed convergence issues. Homotopy continuation, in contrast, holds the potential of a much more comprehensive analysis of the system. For this potential to be realized the computation time needed to employ the method must be brought down significantly. It appears that this will be possible through the use parallel computing and a two part arrangement in which the the most computationally costly portion of the process need only be done once.

BIBLIOGRAPHY

[1] D.J. Bates, I.A. Fotiou, and P. Rostalski. A numerical algebraic geometry approach to nonlinear constrained optimal control. In Decision and Control, 2007 46th IEEE Conference on, pages 6256–6261, 2007.

[2] Y. Uny Cao, Alex S. Fukunaga, and Andrew Kahng. Cooperative mobile robotics: Antecedents and directions. Autonomous Robots, 4(1):7–27, March 1997.

[3] Baoyang Deng, Mihir Sen, and Bill Goodwine. Bifurcations and symmetries of optmial solutions for distributed robotic systems. 2009.

[4] J.P. Desai, J. Ostrowski, and V. Kumar. Controlling formations of multiple mobile robots. In Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on, volume 4, pages 2864–2869 vol.4, 1998.

[5] E. Fiorelli, N.E. Leonard, P. Bhatta, D. Paley, R. Bachmayer, and D.M. Fratantoni. Multi-AUV control and adaptive sampling in monterey bay. In Autonomous Underwater Vehicles, 2004 IEEE/OES, pages 134–147, 2004.

[6] Y. C. Fung. Foundations of Solid Mechanics. Prentice Hall International Series in Dynamics. Prentice Hall, 1977.

[7] S.M. LaValle and S.A. Hutchinson. Optimal motion planning for multiple robots having independent goals. Robotics and Automation, IEEE Transactions on, 14(6):912–925, 1998.

[8] M. Anthony Lewis and Kar-Han Tan. High precision formation control of mobile robots using virtual structures. Autonomous Robots, 4(4):387–403, October 1997.

[9] M. Brett McMickell and Bill Goodwine. Reduction and non-linear controllability of symmetric distributed systems. International Journal of Control, 76(18):1809, 2003.

[10] M. Brett McMickell and Bill Goodwine. Motion planning for nonlinear symmetric distributed robotic formations. The International Journal of Robotics Research, 26(10):1025–1041, October 2007.

[11] Sanford M. Roberts and Jerome S. Shipman. Two-Point Boundary Value Problems: Shooting Methods. Modern Analytic and Computational Methods in Science and Mathematics. American Publishing Company, 1972.

[12] S. Sheikholeslam and C.A. Desoer. Control of interconnected nonlinear dynamical systems: the platoon problem. Automatic Control, IEEE Transactions on, 37(6):806–810, 1992.

[13] Andrew Sommese, Daniel J. Bates, and J. D. Hauenstein. Bertini home page. www.nd.edu/ sommese/bertini, 2009.

[14] Andrew John Sommese and Charles Weldon Wampler. The numerical solution of systems of polynomials arising in engineering and. 2005.

Appendices

APPENDIX A

Code Listing

A.1 Shooting Method

A.1.1 General Nonlinear Shooting Method functions

- Incremental Nonlinear Shooting Method

```
function [Y,T,ICfinal]=nLShootInc(g,dGdy,plotSol,steps,t0,tf,n,r,...
ICind,ICval,ICguess,FCind,FCval,eps,maxIt)
% Guess unknown initial conditions and form initial condition vector
y0 = zeros(n,1);
ind1 = 1;
ind2 = 1;
for i=1:n
    if i == ICind(ind1)
        y0(i) = ICval(ind1);
        if ind1 < r
            ind1 = ind1 + 1;
        end
    else
        y0(i) = ICguess(ind2);
        NoICind(ind2) = i;
        ind2 = ind2 + 1;
    end
end
```

```
% Shoot with initial guess
[T,Y] = ode45(g,[t0 tf],y0);
%plotSol(T,Y);

% Extract final state
yf = Y(length(T),:);

% Calculate intermediate destinations
for i = 1:n-r
    FCvalArray(i,:) =  linspace(yf(FCind(i)),FCval(i),steps);
end

% Step to final destination
for j = 1:steps
    disp('step');
    disp(j);
    [Y,T,ICguess]=nonLinShoot(g,dGdy,plotSol,t0,tf,n,r,...
    ICind,ICval,ICguess,FCind,FCvalArray(:,j),eps,maxIt);
    if ICguess == zeros(length(ICguess),1)
        ICfinal = zeros(length(ICguess),1);
        return;
    end
end
plotSol(T,Y);
ICfinal = ICguess;
end
```

- Nonlinear Shooting Method Code

```
function [Y,T,ICfinal]=nonLinShoot(g,dGdy,plotSol,t0,tf,n,r,ICind,ICval,ICguess,
% [Y,T]=nonLinShoot(g,dgdx,t0,tf,n,r,ICind,ICval,ICguess,FCind,FCval, ...
eps,maxIt)
% Solves the n dimensional nonlinear boundary problem ydot = g(t,y) subject to r
%  initial conditions ICval and n-r initial conditions FCval
```

```
% ARGUMENTS
%   g: Right hand side of the system of differential equation. g(t,y)
%   returns ydot
%   dgdx: Function of the form dGdy(i,j,Y,T). Returns the value of the
%       element of the Jacobian at index (j,i).
%   t0: Initial time
%   tf: Final time
%   n: Dimension of system
%   r: Number of variables for which ICs are specified
%   ICind: Column vector of length r containing the indices of the
%       variables for which initial conditions are specified.
%   ICval: Column vector of length r containing the IC values for the
%       variables specified by ICind.
%   FCind: Column vector of length n-r containing the indices of the
%       variables for which final conditions are specified.
%   FCval: Column vector of length n-r containing the FC values for the
%       variables specified by ICind.
%   eps: Tolerance
%   maxIt: Maximum number of iterations allowed before function times out.

global k;

%debug
global q;
close all

c = 1;
c2 = k*.75;
tArray = linspace(t0,tf,10000);
options = odeset('reltol',1e-4);
EArray = zeros(n-r,1);

xf = zeros(n,1);
NoICind = zeros(n-r,1);
```

```
% Guess unknown initial conditions and form initial condition vector
y0 = zeros(n,1);
ind1 = 1;
ind2 = 1;
for i=1:n
    if i == ICind(ind1)
        y0(i) = ICval(ind1);
        if ind1 < r
            ind1 = ind1 + 1;
    else
        y0(i) = ICguess(ind2);
        NoICind(ind2) = i;
        ind2 = ind2 + 1;
    end
end

% Initialize iteration counter
p = 1;

% Integrate ydot = g(y,t) from t0 to tf
[T,Y] = ode45(g,tArray,y0,options);

%   Calculate error
    E = 0;
    for i = 1:n-r
        EArray(i) = FCval(i)-Y(size(Y,1),FCind(i));
    end
    E = sum(EArray.^2)^0.5;
    Ex = EArray(1:n/4);
    Ey = EArray(n/4+1:n/2);

% DEBUG
disp(E);

while E > eps
```

```
% Initialize arrays for solution
A = zeros(n-r);
b = zeros(n-r,1);

% Initialize adjoint integration counter
for m = 1:n-r
    % Configure final conditions for adjoint equations
    for i = 1:n
        if i == FCind(m)
            xf(i) = 1;
        else
            xf(i) = 0;
        end
    end

    % Integrate adjoint equations from tf to t0
    %[Tx,X] = ode45(@adj,flipud(T),xf);
    [Tx,X] = ode45(@adj,fliplr(tArray),xf,options);

    % Store mth row of left hand matrix (A)

    for j = 1:n-r
        A(m,j) = X(size(X,1),NoICind(j));
    end

    % Compute mth element of right hand vector
    b(m) = FCval(m) - Y(size(Y,1),FCind(m));
end

%   Solve for dy0 (Change in unspecified initial conditions)
dy0 = inv(A)*b;

%   Update initial conditions
for i = 1:n-r
    y0(NoICind(i)) = y0(NoICind(i)) + c*dy0(i);
```

```
    end
    %   Integrate ydot = g(y,t) from t0 to tf
    [T,Y] = ode45(g,tArray,y0,options);

    %DEBUG
    %plotSol(T,Y)

    %   Calculate error
    E = 0;
    for i = 1:n-r
        EArray(i) = FCval(i)-Y(size(Y,1),FCind(i));
    end
    E = sum(EArray.^2)^0.5;
    Ex = EArray(1:n/4);
    Ey = EArray(n/4+1:n/2)

    %   Increment iteration counter
    p = p +1;

    if p > maxIt
        disp('No solution found')
        ICfinal = zeros(length(ICguess),1);
        return;
    end
end

% Output final values of unspecified intitial conditions
ind1 = 1;
ind2 = 1;
ICfinal = zeros(n-r,1);
for i=1:n
    if i ~= ICind(ind1)
        ICfinal(ind2) = Y(1,i);
        ind2 = ind2 + 1;
    else
```

```
        if ind1 < r
            ind1 = ind1 + 1;
        end
    end
end
close all
    function xdot = adj(t,x)
        y = interp1q(T,Y,t)';
        J = dGdy(y,t);
        xdot = -J'*x;
    end
end
```

A.1.2 Functions for the n-robot system

- Main (nRobotMain.m)

```
%clear all
close all
global k;
global n;

%debug
global q;
q = 4;

n = 5;
kArray = [0:2:14 15 15.5:0.1:20];

ICgArray = zeros(length(kArray),2*n);
devArray = zeros(length(kArray), n);
eps = 0.0005;
maxIt = 200;
tstar = 0.25;
```

```
filename = 'IC_04_23_09b.txt';

ICval = zeros(2*n,1);
FCval = zeros(2*n,1);
ICguess = zeros(2*n,1);

for i = 1:n
    ICval(i) = 2*i+4;
    ICval(i+n) = 0;

    FCval(i) = 0;
    FCval(i+n) = 2*i+4;
end
ICguess = [-1,-1,-1,-1,-1,1,1,1,1,1]';

%Generate nominal trajectory
k = 0;
disp(k)
steps = 2;
%[Znom,Tnom,scrap]=nLShootInc(@nRobotsG,@nRobotsJ,@nRobotsPlot,steps,0,1,4*n,2*n
[Znom,Tnom,ICguess]=nLShootInc(@nRobotsG,@nRobotsJ,@nRobotsPlot,steps,0,1,4*n,2*

for i = 1:length(kArray)
    k = kArray(i);
    disp('k')
    disp(k)

    if k < 15
        steps = min(max(2,round(k/2)),20);
    else
        steps = 10;
    end
    [Z,T,ICguess]=nLShootInc(@nRobotsG,@nRobotsJ,@nRobotsPlot,steps,0,1,4*n,2*n,
    devArray(i,:) = measureDev(T,Z,Tnom, Znom, n,tstar);
    ICgArray(i,:) = ICguess';
```

```
    dlmwrite(filename,ICguess', '-append','delimiter','\t','precision',6)
end
figure;
plot(kArray,devArray)
```

- ODE for n-robot system

```
function zdot = nRobotsG(t,z)
    global k;
    global n;

    zdot = zeros(4*n,1);

    for i = 1:2*n
        zdot(i) = z(i+2*n);
    end



    zdot(2*n+1) = -h( z(1), z(2) , z(n+1), z(n+2) );
    %zdot(2*n+1) = h( z(1), z(2) , z(n+1), z(n+2) );

    for i = 2*n+2:3*n-1
        zdot(i) = h( z(i-2*n-1), z(i-2*n) , z(i-n-1), z(i - n) )...
                    -h( z(i-2*n), z(i-2*n+1), z(i-n), z(i-n+1) );
    end

    zdot(3*n) = h( z(n-1), z(n), z(2*n-1), z(2*n) );
    %zdot(3*n) = -h( z(n-1), z(n), z(2*n-1), z(2*n) );

    zdot(3*n+1) = -h( z(n+1), z(n+2), z(1), z(2) );
    %zdot(3*n+1) = h( z(n+1), z(n+2), z(1), z(2) );

    for i = 3*n+2:4*n-1
        zdot(i) = h(z(i-2*n-1), z(i-2*n), z(i-3*n-1), z(i-3*n) )...
                    -h(z(i-2*n), z(i-2*n+1), z(i-3*n), z(i-3*n+1) );
```

```
    end

    zdot(4*n) = h( z(2*n-1), z(2*n), z(n-1), z(n) );
    %zdot(4*n) = -h( z(2*n-1), z(2*n), z(n-1), z(n) );

    % Crazy debug
    %zdot = -zdot;
end
```

- Jacobian for n-robot system

```
function J = nRobotsJ(z,t)
global k;
global n;

J = zeros(4*n);

for i = 1:2*n
    for j = 1:4*n
        if j == i+2*n
            J(i,j) = 1;
        else
            J(i,j) = 0;
        end
    end
end

i = 2*n+1;
for j = 1:4*n
    if j == i-2*n
        J(i,j) = -k*f1( z(j), z(j+1), z(j+n), z(j+n+1) );

    elseif j == i-2*n+1
        J(i,j) = k*f1( z(j-1), z(j), z(j+n-1), z(j+n) );
```

```
    elseif j == i-n
        J(i,j) = -k*f2( z(j-n), z(j-n+1), z(j), z(j+1) );

    elseif j == i-n+1
        J(i,j) = k*f2( z(j-n-1), z(j-n), z(j-1), z(j) );

    else
        J(i,j) = 0;
    end
end

for i = 2*n+2:3*n-1
    for j=1:4*n
        if j == i-2*n-1
            J(i,j) = k*f1( z(j), z(j+1), z(j+n), z(j+n+1) );

        elseif j == i-2*n
            J(i,j) = -k*( f1( z(j-1), z(j), z(j+n-1), z(j+n) )...
                          + f1( z(j), z(j+1), z(j+n), z(j+n+1) ) );

        elseif j == i-2*n+1
            J(i,j) = k*f1( z(j-1), z(j), z(j+n-1), z(j+n) );

        elseif j == i-n-1
            J(i,j) = k*f2( z(j-n), z(j-n+1), z(j), z(j+1) );

        elseif j == i-n
            J(i,j) = -k*( f2( z(j-n-1), z(j-n), z(j-1), z(j) )...
                          + f2( z(j-n), z(j-n+1), z(j), z(j+1) ) );

        elseif j == i-n+1
            J(i,j) = k*f2( z(j-n-1), z(j-n), z(j-1), z(j) );

        else
            J(i,j) = 0;
```

```
        end
    end
end

i = 3*n;

for j = 1:4*n
    if j == i-2*n-1
        J(i,j) = k*f1( z(j), z(j+1), z(j+n), z(j+n+1) );

    elseif j == i-2*n
        J(i,j) = -k*f1( z(j-1), z(j), z(j+n-1), z(j+n) );

    elseif j == j-n-1
        J(i,j) = k*f2( z(j-n), z(j-n+1), z(j), z(j+1) );

    elseif j == j-n
        J(i,j) = -k*f2( z(j-n-1), z(j-n), z(j-1), z(j) );
    else
        J(i,j) = 0;
    end
end

% y
i = 3*n+1;
for j = 1:4*n
    if j == i-3*n
        J(i,j) = -k*f2( z(j+n), z(j+n+1), z(j), z(j+1) );

    elseif j == i-3*n+1
        J(i,j) = k*f2( z(j+n-1), z(j+n), z(j-1), z(j) );

    elseif j == i-2*n
        J(i,j) = -k*f1( z(j), z(j+1), z(j-n), z(j-n+1) );
```

```
    elseif j == i-2*n+1
        J(i,j) = k*f1( z(j-1), z(j), z(j-n-1), z(j-n) );

    else
        J(i,j) = 0;
    end
end

for i = 3*n+2:4*n-1
    for j=1:4*n
        if j == i-3*n-1
            J(i,j) = k*f2( z(j+n), z(j+n+1), z(j), z(j+1) );

        elseif j == i-3*n
            J(i,j) = -k*( f2( z(j+n-1), z(j+n), z(j-1), z(j) )...
                          + f1( z(j+n), z(j+n+1), z(j), z(j+1) ) );

        elseif j == i-3*n+1
            J(i,j) = k*f2( z(j+n-1), z(j+n), z(j-1), z(j) );

        elseif j == i-2*n-1
            J(i,j) = k*f1( z(j), z(j+1), z(j-n), z(j-n+1) );

        elseif j == i-2*n
            J(i,j) = -k*( f1( z(j-1), z(j), z(j-n-1), z(j-n) )...
                          + f1( z(j), z(j+1), z(j-n), z(j-n+1) ) );

        elseif j == i-2*n+1
            J(i,j) = k*f1( z(j-1), z(j), z(j-n-1), z(j-n) );

        else
            J(i,j) = 0;

        end
```

```
    end
end

i = 4*n;

for j = 1:4*n
    if j == i-3*n-1
        J(i,j) = k*f2( z(j+n), z(j+n+1), z(j), z(j+1) );

    elseif j == i-3*n
        J(i,j) = -k*f2( z(j+n-1), z(j+n), z(j-1), z(j) );

    elseif j == j-2*n-1
        J(i,j) = k*f1( z(j), z(j+1), z(j-n), z(j-n+1) );

    elseif j == j-2*n
        J(i,j) = -k*f1( z(j-1), z(j), z(j-n-1), z(j-n) );
    else
        J(i,j) = 0;
    end
end

end
```

- Calculation of d_{dev}

```
function dev = measureDev(T, Z,Tnom,Znom,n,tstar)
    %[C,index] = min(abs(T-tstar));
    dev = zeros(1,n);

    %Calculate distance to straightline path
    for i = 1:n
        % Position at t = tstar
        x = interp1q(T,Z(:,i),tstar);
        y = interp1q(T,Z(:,n+i),tstar);
```

```
        % Straight line position at t = tstar
        xo = interp1q(Tnom,Znom(:,i),tstar);
        yo = interp1q(Tnom,Znom(:,n+i),tstar);
        % Deviation as distance between points
        dev(i) = sqrt((x-xo)^2 + (y-yo)^2);
        % Add sign
        if y<2*i+4-x
            dev(i) = -dev(i);
        end
    end
end
```

A.2 Finite Difference Method

- Main (FDMmain.m)

```
clear all
close all

global h
global k
global dbar
global A
global B

epsilon = 0.0001;
lambda = .05;
n = 5;
m = 20;
T = linspace(0,1,m+1);
h = abs(T(2)-T(1));
kArray = 0:.1:10;
devArray = zeros(length(kArray),n);
dbar = 2;
A = [[0:n-1]*2 + 6,zeros(1,n)]';
```

```
B = [zeros(1,n),[0:n-1]*2 + 6]';

Z0 = zeros(m+1,2*n);

for i = 1:n
    Z0(:,i) = -T*A(i)+A(i);
    Z0(:,n+i) = T*B(n+i);
end

for i = 1:length(kArray)
    k = kArray(i)
    Z = newton(@Fn,@Jacobian,Z0,lambda,epsilon);
    devArray(i,:) = measureDev(T,Z,n,0.25);
    Z0 = Z;
end

figure
plot(kArray,devArray)
```

- Newton's Method Code

```
function x = newton(fun,Jfun,x0,lambda0,eps)
E = eps+1;
x = x0;
count = 1;
lambda = lambda0;
while E >eps
    xnew = x - reshape(lambda*(Jfun(x)\fun(x)),size(x0,1),size(x0,2));
    Enew = sum(sum((xnew-x).^2))^0.5;
    E = Enew;
    x = xnew;
    count = count+1;
    if count >= 500
        break
    end
```

```
end
end
```

- Finite Difference Approximation for n-robot system

```
function Fout = Fn(Z)
global h
global k
global dbar
global A
global B
% Evaluates the F system of equations at Z

% Find number of robots and number of mesh points
n = size(Z,2)/2;
m = size(Z,1)-1;

% Initialize output array
F = zeros(2*n,m+1);

% Evaluate components corresponding to boundary conditions
for i = 1:2*n
    F(i,1) = Z(1,i)-A(i);
    F(i,m+1) = Z(m+1,i)-B(i);
end

% Evaluate components corresponding to interior mesh points
for j = 2:m
    z = Z(j,:);
    d = dist([z(1),z(2),z(n+1),z(n+2)]);
    F(1,j) = zDDApprox(Z,h,1,j) + k*(z(2)-z(1))*(d-dbar)/d;
    F(n+1,j) = zDDApprox(Z,h,n+1,j) + k*(z(n+2)-z(n+1))*(d-dbar)/d;

    for i = 2:n-1
        dplus = dist([z(i),z(i+1),z(n+i),z(n+i+1)]);
```

```
        dminus = dist([z(i-1),z(i),z(n+i-1),z(n+i)]);
        F(i,j) = zDDApprox(Z,h,i,j)...
                - k*( (z(i)-z(i-1))*(dminus-dbar)/dminus...
                        -(z(i+1)-z(i))*(dplus-dbar)/dplus);
        F(n+i,j) =  zDDApprox(Z,h,n+i,j)...
                    - k*( (z(n+i)-z(n+i-1))*(dminus-dbar)/dminus...
                            -(z(n+i+1)-z(n+i))*(dplus-dbar)/dplus);
    end

    d = dist([z(n-1),z(n),z(2*n-1),z(2*n)]);
    F(n,j) = zDDApprox(Z,h,n,j) - k*(z(n)-z(n-1))*(d-dbar)/d;
    F(2*n,j) = zDDApprox(Z,h,2*n,j) - k*(z(2*n)-z(2*n-1))*(d-dbar)/d;
end

Fout = reshape(F',2*n*(m+1),1);
end
```

- Jacobian of Finite Difference Approximation of n-Robot System

```
function J = Jacobian(Z)
global h
global k
global dbar
% Computes the Jacobian of the F system of equations

% Find number of robots and number of mesh points
n = size(Z,2)/2;
m = size(Z,1)-1;

J = zeros(2*n*(m+1));

L = rowIndex(n,m); %Returns a 8 x (m-1) x (2n+1) array of row indices

for b = 1:m-1
    z = Z(b,:);
```

```
for i = 1:4
    l = L(i,b,1);
    p = l;
    switch i
        case 1
            J(l,p) = -2/h^2 + k*psi1([z(1),z(2),z(n+1),z(n+2)],dbar);%2)
        case 2
            J(l,p) = -2/h^2 + k*psi1([z(n+1),z(n+2),z(1),z(2)],dbar);
        case 3
            J(l,p) = -2/h^2 + k*psi1([z(n-1),z(n),z(2*n-1),z(2*n)],dbar);%6)
        case 4
            J(l,p) = -2/h^2 + k*psi1([z(2*n-1),z(2*n),z(n-1),z(n)],dbar);%7)
    end

    p = l + n*(m+1);
    switch i
        case 1
            J(l,p) = k*psi3([z(1),z(2),z(n+1),z(n+2)],dbar);%13)
        case 3
            J(l,p) = k*psi3([z(n-1),z(n),z(2*n-1),z(2*n)],dbar);%15)
    end

    p = l - n*(m+1);
    switch i
        case 2
            J(l,p) = k*psi3([z(n+1),z(n+2),z(1),z(2)],dbar);%18)
        case 4
            J(l,p) = k*psi3([z(2*n-1),z(2*n),z(n-1),z(n)],dbar);%20)
    end
end

% Alpha regime 1
a = 1;
    for i = 5:6
        l = L(i,b,a);
```

```
        p = l;
        J(l,p) = 1;%1)
    end

    l = L(7,b,a);
    p = l+1;
    J(l,p) = 1/h^2; %8)
    p = l-1;
    J(l,p) = 1/h^2; %8)
    p = l+(m+1);
    J(l,p) = -k*psi1([z(a),z(a+1),z(n+a),z(n+a+1)],dbar); %9)
    p = l + (n+1)*(m+1);
    J(l,p) = -k*psi3([z(a),z(a+1),z(n+a),z(n+a+1)],dbar); %16)

    l = L(8,b,a);
    p = l+(m+1);
    J(l,p) = -k*psi1([z(n+a),z(n+a+1),z(a),z(a+1)],dbar); %10)
    p = l-(n-1)*(m+1);
    J(l,p) = -k*psi3([z(n+a),z(n+a+a),z(a),z(a+1)],dbar); %21)

% Alpha Regime 2
for a = 2:n-1
    for i = 5:6
        l = L(i,b,a);
        p = l;
        J(l,p) = 1;%1)
    end


    l = L(7,b,a);
    p = l;
    J(l,p) = -2/h^2 + k*( psi1([z(a-1),z(a),z(n+a-1),z(n+a)],dbar)...
                          + psi1([z(a),z(a+1),z(n+a),z(n+a+1)],dbar) );%4)
    p = l+1;
    J(l,p) = 1/h^2; %8)
```

```
    p = l-1;
    J(l,p) = 1/h^2; %8)
    p = l+(m+1);
    J(l,p) = -k*psi1([z(a),z(a+1),z(n+a),z(n+a+1)],dbar); %9)
    p = l-(m+1);
    J(l,p) = -k*psi1([z(a-1),z(a),z(n+a-1),z(n+a)],dbar); %11)
    p = l + n*(m+1);
    J(l,p) = k*( psi3([z(a-1),z(a),z(n+a-1),z(n+a)],dbar) ...
                + psi3([z(a),z(a+1),z(n+a),z(n+a+1)],dbar) );%14)
    p = l + (n+1)*(m+1);
    J(l,p) = -k*psi3([z(a),z(a+1),z(n+a),z(n+a+1)],dbar); %16)
    p = l + (n-1)*(m+1);
    J(l,p) = -k*psi3([z(a-1),z(a),z(n+a-1),z(n+a)],dbar); %17)


    l = L(8,b,a);
    p = l;
    J(l,p) = -2/h^2 + k*( psi1([z(n+a-1),z(n+a),z(a-1),z(a)],dbar)...
                              + psi1([z(n+a),z(n+a+1),z(a),z(a+1)],dbar) );%5)
    p = l+(m+1);
    J(l,p) = -k*psi1([z(n+a),z(n+a+1),z(a),z(a+1)],dbar); %10)
    p = l-(m+1);
    J(l,p) = -k*psi1([z(n+a-1),z(n+a),z(a-1),z(a)],dbar); %12)
    p = l - n*(m+1);
    J(l,p) = k*( psi3([z(n+a-1),z(n+a),z(a-1),z(a)],dbar) ...
                + psi3([z(n+a),z(n+a+1),z(a),z(a+1)],dbar) );%19)
    p = l-(n-1)*(m+1);
    J(l,p) = -k*psi3([z(n+a),z(n+a+1),z(a),z(a+1)],dbar); %21)
    p = l-(n+1)*(m+1);
    J(l,p) = -k*psi3([z(n+a-1),z(n+a),z(a-1),z(a)],dbar); %22)
end
% Alpha Regime 3
a = n;
    for i = 5:6
        l = L(i,b,a);
```

```
        p = l;
        J(l,p) = 1;%1)
    end


    l = L(7,b,a);
    p = l+1;
    J(l,p) = 1/h^2; %8)
    p = l-1;
    J(l,p) = 1/h^2; %8)
    p = l-(m+1);
    J(l,p) = -k*psi1([z(a-1),z(a),z(n+a-1),z(n+a)],dbar); %11)
    p = l + (n-1)*(m+1);
    J(l,p) = -k*psi3([z(a-1),z(a),z(n+a-1),z(n+a)],dbar); %17)


    l = L(8,b,a);
    p = l-(m+1);
    J(l,p) = -k*psi1([z(n+a-1),z(n+a),z(a-1),z(a)],dbar); %12)
    p = l-(n+1)*(m+1);
    J(l,p) = -k*psi3([z(n+a-1),z(n+a),z(a-1),z(a)],dbar); %22)

% Alpha Regime 4
for a = n+1:2*n
    for i = 5:6
        l = L(i,b,a);
        p = l;
        J(l,p) = 1;%1)
    end

    l = L(7,b,a);
    p = l+1;
    J(l,p) = 1/h^2; %8)
    p = l-1;
    J(l,p) = 1/h^2; %8)
```

```
    end
end

%disp(min(svd(J)))
end
```

A.3 Homotropy Method

- Perl Script to Create Bertini Input File

```
#!/usr/bin/perl
# Author: Bill Goodwine
# program to make the input file for the robotic unicycle system

$k = 10;                  # weighting paramter
$n = 5;                     # number of robots
$m = 2;                     # number of time discretization points


open OUT, "> input";

print OUT "%robots = $n, points = $m, k = $k\n\n";

print OUT "CONFIG\n\n";
print OUT "USEREGENERATION: 1;\n";
print OUT "MPTYPE: 2;\n\n";
print OUT "SECURITYMAXNORM: 1e8;\n";
print OUT "MAXNORM: 1e8;\n\n";
print OUT "TRACKTOLBEFOREEG: 1e-6;\n";
print OUT "TRACKTOLDURINGEG: 1e-6;\n";
print OUT "FINALTOL: 1e-12;\n\n";
print OUT "SLICETOLBEFOREEG: 1e-6;\n";
print OUT "SLICETOLDURINGEG: 1e-6;\n";
print OUT "SLICEFINALTOL: 1e-12;\n\n";
print OUT "COEFFBOUND: ",100,";\n";
```

```
print OUT "DEGREEBOUND: 3;\n\n";
print OUT "ODEPREDICTOR: 2;\n";
print OUT "SAMPLEFACTOR: 0.1;\n\n";
print OUT "END;\n\n";

print OUT "INPUT\n\n";

print OUT "variable_group ";
for($i=1;$i<=$n-1;$i++) {
  for($j=1;$j<=$m-1;$j++) {
    print OUT "xi$i\j$j, ";
    print OUT "yi$i\j$j, ";
    print OUT "di$i\j$j, ";
  }
}

for($j=1;$j<=$m-2;$j++) {
  print OUT "xi$n\j$j, ";
  print OUT "yi$n\j$j, ";
}

print OUT "xi$n\j",$m-1,", ";
print OUT "yi$n\j",$m-1,";\n\n";

for($i=1;$i<=$n;$i++) {
  print OUT "constant xi$i\j0,xi$i\j$m,yi$i\j0,yi$i\j$m;\n"
}

print OUT "constant dbar,k,h;\n\n";

for($i=1;$i<=$n-1;$i++) {
  for($j=1;$j<=$m-1;$j++) {
    print OUT "function fi$i\j$j, gi$i\j$j;\n";
  }
}
```

```
for($j=1;$j<=$m-1;$j++) {
  print OUT "function fi$n\j$j, gi$n\j$j;\n";
}

for($i=1;$i<=$n-1;$i++) {
  for($j=1;$j<=$m-1;$j++) {
    print OUT "function hi$i\j$j;\n";
  }
}

print OUT "\n";
for($i=1;$i<=$n;$i++) {
  print OUT "xi$i\j0 = ",4+2*$i,";\n";
  print OUT "xi$i\j$m = ",0,";\n";
  print OUT "yi$i\j0 = ",0,";\n";
  print OUT "yi$i\j$m = ",4+2*$i,";\n";
}

print OUT "dbar = 2;\n";
print OUT "k = $k;\n";
print OUT "h = ",1/$m,";\n\n";


for($j=1;$j<=$m-1;$j++) {
  print OUT "fi1\j$j = (xi1\j",$j+1,"-2*xi1\j$j+xi1\j",$j-1,")*di1\j$j-h^2*k*((x
  print OUT "gi1\j$j = (yi1\j",$j+1,"-2*yi1\j$j+yi1\j",$j-1,")*di1\j$j-h^2*k*((y
}

for($i=2;$i<=$n-1;$i++) {
  for($j=1;$j<=$m-1;$j++) {
    print OUT "fi$i\j$j = (xi$i\j",$j+1,"-2*xi$i\j$j+xi$i\j",$j-1,")*di$i\j$j*di
    print OUT "gi$i\j$j = (yi$i\j",$j+1,"-2*yi$i\j$j+yi$i\j",$j-1,")*di$i\j$j*di
  }
}
```

```
for($j=1;$j<=$m-1;$j++) {
  print OUT "fi$n\j$j = (xi$n\j",$j+1,"-2*xi$n\j$j+xi$n\j",$j-1,")*di",$n-1,"\j$
  print OUT "gi$n\j$j = (yi$n\j",$j+1,"-2*yi$n\j$j+yi$n\j",$j-1,")*di",$n-1,"\j$
}

for($j=1;$j<=$m-1;$j++) {
  print OUT "hi1\j$j = di1\j",$j,"^2 - (xi1\j$j-xi",1+1,"\j$j)^2 - (yi1\j$j-yi",
}


for($i=2;$i<=$n-1;$i++) {
  for($j=1;$j<=$m-1;$j++) {
    print OUT "hi$i\j$j = di$i\j",$j,"^2 - (xi$i\j$j-xi",$i+1,"\j$j)^2 - (yi$i\j
  }
}

print OUT "\nEND;\n\n";
close OUT;
```