Particle Swarm Optimization Applied to the Design of a Nonlinear Control

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

David J. Broderick

Certificate of Approval:

A. Scottedward Hodel
Associate Professor
Department of Electrical and Computer Engineering

John Y. Hung, Chair
Associate Professor
Department of Electrical and Computer Engineering

Thomas S. Denney
Associate Professor
Department of Electrical and Computer Engineering

Stephen L. McFarland
Dean, Graduate School

Particle Swarm Optimization Applied to the Design of a Nonlinear Control

David J. Broderick

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
May 11, 2006

Particle Swarm Optimization Applied to the Design of a Nonlinear Control

David J. Broderick

Signature of Author

Date

Copy sent to:

Name Date

Vita

David was born on May 25, 1979 in Eastern Connecticut to Jack and Gisela Broderick. After attending the Kingswood-Oxford School in West Hartford, Connecticut he continued his education at the University of Hartford. While working in the insurance industry as a software developer and tester, he ultimately earned his B.S. degree from Hartford. An internship with the National Aeronautics and Space Administration led David to Alabama where he has pursued graduate work at Auburn University. His time at Auburn has proven formative both personally and intellectually.

Thesis Abstract

Particle Swarm Optimization Applied to the Design of a Nonlinear Control

David J. Broderick

Master of Science, May 11, 2006
(B.S., University of Hartford, 2003)

77 Typed Pages

Directed by John Y. Hung

A method of searching for a tuning of an input to state linearizing controller is presented. The problem of finding the appropriate weights of the control law's terms is treated as an optimization problem. Given the highly nonlinear surfaces that are likely to be searched, particle swarm optimization is applied. The MEMS parallel plate actuator is used to explore the effectiveness of this technique. The resulting tuning of the controller is then compared to that of the analytically designed solution.

Acknowledgments

I must begin by thanking my family- Jack, Gisela, and Daniel- for their support in all of its forms. They have encouraged me throughout my education and ensured that I have had access to anything I have needed to pursue my goals. Whether it was in the form of tutoring in my early years, financial support later in my education, or just listening to me talk through a problem over the telephone all of them have helped me achieve success.

I must also thank Dr. John Y. Hung, my advisor, without whom I would not have been able to complete my degree. Both inside the classroom and out, Dr. Hung displayed not only technical competence but, more importantly, a caring patience.

I would like to thank the other members of my committee, Dr. Thomas S. Denney, and Dr. A Scottedward Hodel for their support and advice throughout my time at Auburn University.

Style manual or journal used Bibliography conforms to those in the IEEE Transactions.

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`, MATLAB, and SIMULINK.

Table of Contents

List of Figures x

1 Introduction 1
1.1 Past Applications of Particle Swarm Optimization to Linear Control . . . 1
1.2 Particle Swarm Optimization for Nonlinear Control 2

2 Particle Swarm Optimization 3
2.1 Original Implementation 3
2.2 Gaussian Particle Swarm 6
2.3 Nature of Population-based Algorithms 7
2.4 Swarm Metrics 8
2.4.1 Global Best Solution 9
2.4.2 Standard Deviation in $\Re^m$ 9
2.4.3 Distance Moved by Global Best Solution 11
2.5 Typical Behavior of Metrics on a Well-behaved Surface 12

3 Application to Nonlinear Control 16
3.1 Plant Model 16
3.2 Review of MEMS Actuator Control Methods 18
3.3 Analytical Solution 20
3.4 Selection of Desired Dynamics 21
3.5 Implementation Issues 24
3.6 Evaluation of Solutions 25

4 Resultant Tuning and Performance 28
4.1 Response Comparison 28
4.2 Behavior of Metrics 31
4.3 Parameter Convergence 32
4.4 Model Limitations 34

5 Conclusions and Future Work 37
5.1 The Good 37
5.2 The Bad 37
5.3 The Ugly 38

Bibliography 39

Appendices 41

A Code Listing used in Simulations 42
A.1 Plotting Code for Swarm Test Surface 42
A.2 Implementation of Gaussian Particle Swarm for Testing 42
A.3 Evaluation Function for Swarm Testing 44
A.4 Implementation of Gaussian Particle Swarm for Parallel Plate Actuator . 45
A.5 Evaluation function used for Parallel Plate Actuator Problem 48
A.6 Model reference SIMULINK model file 48
A.7 Standard Deviation Function 67

List of Figures

2.1 Test surface used for metrics demonstration 13

2.2 Global best value of swarm on test surface 14

2.3 Standard deviation of swarm on test surface 15

2.4 Distance moved by global best solution on test surface 15

3.1 Parallel Plate Actuator 17

3.2 Model reference system used in evaluating candidate solutions. 26

3.3 Model of linearizing controller using particle location as input. 27

3.4 Model of the parallel plate actuator used in simulation. 27

4.1 Response of system with analytically designed control. 29

4.2 Best tuning of the controller found by swarm. 30

4.3 Global best value of swarm while tuning linearizing control. 32

4.4 Standard deviation of swarm while tuning linearizing control. 33

4.5 Distance moved by global best solution while tuning linearizing control. . 33

4.6 Best tuning of the controller with an inappropriate reference signal. . . . 36

Chapter 1

Introduction

Particle Swarm Optimization (PSO) has become a widely used optimization technique given the simplicity of its implementation and the quality of its results. However, the application of PSO in the field of controls has been mostly limited to the tuning of linear controllers.

1.1 Past Applications of Particle Swarm Optimization to Linear Control

PSO is useful in the adjustment of continuous parameters which makes it an attractive means for tuning a PID controller. Liu et al.[1] examine the problem of optimization of a PID and compare the results to that of traditional tuning methods such as Ziegler-Nichols. Gaing[2] applies this approach to PID tuning in the control of a linearized model of an automatic voltage regulator. The additional step is taken to examine how the tuning process performs compared to that of a PID controller tuned with a genetic algorithm (GA). It is found that the PSO tuning out performs that of the GA tuning with regard to the number of iterations taken to reach an acceptable result. In [3], Hu et al. examine a different application of this technique by applying it to a servo control problem. Again, the results are compared to that of a GA based tuning method with similar results.

In [4], Zheng et al. further the algorithm's purpose by developing a robust PID controller based on PSO. While this is a novel application of PSO in the area of controls,

it depends on the linear control paradigm and suffers from the inherent difficulties of linear techniques.

1.2 Particle Swarm Optimization for Nonlinear Control

The aim of the work presented here is to extend the application of PSO to the tuning of nonlinear controllers. To demonstrate this on one type of nonlinear control, an input to state linear controller is used, under the assumption that the general structure of the control law is known. The law takes the form of a weighted sum of individual terms. The tuning process accounts for uncertainty in the model parameters and can be later extended to encompass controllers whose structure is not well known.

The concept presented is then applied to the MEMS Parallel Plate Actuator. (PPA) The actuator depends on electrostatic force to attract the two plates together. The natural relaxation of the actuator is used to return the moving plate to its nominal position. The analytically designed input to state linearizing control is then compared to that of a PSO designed control.

Chapter 2

Particle Swarm Optimization

Kennedy and Eberhart developed a novel optimization technique in 1995 [5] that proved adept at efficiently locating an optimal solution on nonlinear surfaces. Particle Swarm Optimization (PSO), as originally proposed, is a population-based algorithm whose concept is rooted in the modeling of social information sharing. While it is a novel approach to optimization PSO does share some similarities to genetic algorithms and programming in concept and behavior. The original intent was to study the nature of information sharing in flocks of birds, schools of fish, or groups of people where the collaborative intelligence of the group was used to benefit each individual.

2.1 Original Implementation

PSO is an iterative process which in each iteration evaluates the solutions represented by the particle locations and adjusts the particles' velocities based on prior knowledge. The memory capabilities of the swarm take the form of a global best solution and personal best solutions. The global best solution represents the most favorably evaluated particle location by any particle in the population. This global memory gives the swarm its social intelligence and allows information sharing among particles.

In addition to the social component of the swarm's memory, each individual particle in the population maintains a personal best solution. This personal best solution represents the most favorably evaluated location that the particle has visited. Including this memory in the algorithm aids in avoiding premature convergence on a local optimum.

The first step in each iteration is the evaluation of each solution. The means of evaluation is specific to each problem and effects not only the ability of the swarm to search effectively but also the optimum solution that the swarm will converge on. After evaluation of each particle is completed current scores are compared against the personal best solutions and retained in memory if they have proven to offer a better solution. Each of these personal best scores is then compared against the global score to determine if a new global best has been found.

The core concept of the PSO algorithm is the calculation of particle velocity which takes place after the evaluations of current solutions, or locations, is completed. Equation 2.1 shows how the velocity of each particle is determined based on three terms. The nomenclature used for the velocity equations is found in Table 2.1. The first term represents the inertia of a particle. Part of the current iteration's velocity is dependent on the previous velocity of the particle. This term, in the original implementation of PSO, allows a particle to develop momentum which will often carry it through local optima before decelerating when a global optima is found. It is this ability which makes PSO well suited to highly nonlinear surfaces with many local optima. This term has been appropriately named the inertial term.

$$v_t = \underbrace{v_{t-1}}_{inertia} + \underbrace{w_c\gamma_1(b_{pij} - p_{ij})}_{cognitive\ term} + \underbrace{w_s\gamma_2(b_{gi} - p_{ij})}_{social\ term} \tag{2.1}$$

The next term, the cognitive term, represents the particle's tendency to rely on its own past experiences when chosing a direction to search. The particle's current location and the location of its personal best solution are used to determine the direction and the magnitude of this component of the velocity calculation. The cognitive term is also

Table 2.1: Nomenclature used for PSO velocity and metrics

Variable	Description
v_t	Velocity of a particle for a given iteration, t
p_{ij}	Location of a single particle, j, on a single dimension, i
b_{pij}	Personal best location for a given dimension and particle, p_{ij}
b_{gi}	Global best location on a given dimension, i
γ	Uniformly Random number from 0 to 1
ζ	Random number with Gaussian distribution
σ_t	Standard deviation of the swarm at iteration t
$\bar{p}_i$	Mean value of the swarm on dimension i

weighted by a uniformly random number inclusively between zero and two. The range of the random factor of the term allows for a particle to overfly or underfly the best solution and therefore searching additional locations for better solutions.

The third, and final, term represents the collective tendency of the swarm to share the global best solution and use this social knowledge in deciding a direction to search. Its structure is similar to that of the cognitive term with the exception of the global best solution's location replacing the personal best solution location. The same random factor is used for similar reasons. The social term, as it has been named, pulls the particles toward the current global optima allowing for the swarm to converge to a single location. The last two terms are weighted by uniformly distributed random numbers, γ_1 and γ_2, which range from 0 to 1 and are typically then scaled by weights w_c and w_s set to 2.

2.2 Gaussian Particle Swarm

A strong emphasis is placed on tuning the swarm's weights when discussing the algorithms performance on a particular surface. Krohling has addressed this issue [6] by altering the velocity equation to appear as in equation 2.2.

$$v_t = \underbrace{v_{t-1}}_{inertia} + \underbrace{|\zeta_1|\,(b_{pij} - p_{ij})}_{cognitive\ term} + \underbrace{|\zeta_2|\,(b_{gi} - p_{ij})}_{social\ term} \tag{2.2}$$

The original implementation weights each term with a uniformly random number between 0 and the weight, which is typically 2. Kennedy and Eberhart designed the original algorithm in such a way that a particle's location would have an equal opportunity of overshooting the current best solution as it would to search the space between it and the best solution.

Using Gaussian PSO, the weights of the social and cognitive terms are replaced with random numbers with a gaussian distribution, ζ_1 and ζ_2. To ensure that a particle is always moving toward the best solution the absolute value of the distribution is taken so that the random weight is always positive. After this change, a particle is more likely to search the space between its current location and the best location. It is less likely to overshoot the best location by twice the distance but it is still possible. Finally, using this distribution allows for a particle to ocassionally make a large jump in location. It is this last ability that gives this implementation of the algorithm its greater ability to escape local optima and therefore speed convergence to the global optimum. Krohling also observed that the standard deviation of the gaussian distribution was greater that that of the uniform distribution allowing for greater variability in the random factors of the terms.

Krohling is left to experimentally prove that his approach provides superior performance than Kennedy and Eberhart's original implementation. This was accomplished using a number of test functions and a static tuning of the original algorithm as a benchmark with both terms weighted by 2. Further manual tuning may result in better performance from the original algorithm but removing the need for tuning in the first place and offering better general performance without the need for many test runs makes the gaussian algorithm attractive.

2.3 Nature of Population-based Algorithms

Since PSO is classified as a population based optimization algorithm it is necessary to examine the similarities and differences between it and Genetics Algorithms (GAs), the more pervasive of this type of algorithm.

Both PSO and GAs operate to find a better solutions as they iterate, GAs through generations and PSO through locations of particles. However, with both algorithms there is no guarantee that after a given set of conditions are met that a global optimum has been located. Therefore, the solution determined by the techniques is not said to be optimal but rather near-optimal.

John Holland, who first developed GAs, introduced the Schema Theorem. This statement is often used not as proof but rather an explanation of the nature of GA solutions.

> "Short schema with better than average cost occur exponentially more frequently in the next generation. Schema with costs worse than average occur less frequently in the next generation."

A schema is used here to designate a genetic representation of a solution much like a particle's location represents a solution. This approach leads to the optimization of relative scores and not absolute scores. In other words, the near-optimal value found by both algorithms can also be termed a relative optimal value that may or may not be the absolute optimal value. If the region being searched is not well known there is no method of determining the relationship between the two values.

Both algorithms suffer from the need for extensive simulation in order to evaluate candidate solutions. PSO and GAs must balance the need for efficient computation of a solution and the need to obtain a richer evaluation though more intensive simulation. If the applications of the algorithms are not time sensitive this is not as critical an issue. If the algorithms are to be used in a time sensitive application more attention must be payed to computational efficiency at the risk of not always finding an optimal solution.

2.4 Swarm Metrics

Simple surfaces of two dimensions or fewer allow for easy visualization during the search process. Surfaces of greater dimensionality do not allow easy visualization and therefore require a set of metrics to be defined to offer insight into the status of the swarm during simulation. The measurements used in this study, as defined below, are the global best solution, standard deviation, and distance moved by the global best solution. All of these values are observed over time as the swarm is allowed to iterate through its search process.

2.4.1 Global Best Solution

The global best solution is not only a key value in calculating the velocity of each particle, but serves well as a means of observing the progression of the search. On a simple surface such as the cornfield example used in [5] the global best solution will converge on the global optimum quickly and smoothly. On more complex, nonlinear surfaces the global best solution can get caught in local optima over several iterations before breaking out to continue the search for the global optimum.

2.4.2 Standard Deviation in $\Re^m$

The strength of PSO is found in the spatial diversity of the individual particles. As such, a numerical measurement was sought to determine how spread out the swarm is during any given iteration. The field of statistics offers standard deviation as a means to quantify the distribution of sampled data. While this application of standard deviation does not measure the spread of sampled data, it does act as an effective indicator of spatial diversity.

Examining the classic representation of standard deviation, which operates in $\Re^1$, as shown in equation 2.3, it is necessary to adapt the concept to the problem at hand. Calculating standard deviation requires the use of the mean value. In $\Re^1$ this is a simple average, the summation of the values divided by the number of values. In $\Re^m$, the mean value is the summation of the m x 1 vectors representing the particle locations divided by the population of the swarm, n. This calculation is detailed in equation 2.4.

$$\sigma = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(p_j - \bar{p})^2} \tag{2.3}$$

$$\bar{\mathbf{p}} = \frac{1}{n}\sum_{j=1}^{n} \mathbf{p_j} \tag{2.4}$$

Variance is defined as the square of the distance from a single particle to the mean value. Using the representation of particle and the mean value as shown in equation 2.5 the distance of the *jth* particle to the mean is found using equation 2.6. In general terms, the standard deviation is the square root of the average variance. Squaring equation 2.6 results in the variance in $\Re^m$ which replaces the variance in the one dimensional standard deviation formula. The resulting expression for standard deviation in m dimensions is equation 2.7.

$$\mathbf{p_j} = \begin{bmatrix} p_{1j} \\ p_{2j} \\ \vdots \\ p_{mj} \end{bmatrix} \qquad \bar{\mathbf{p}} = \begin{bmatrix} \bar{p}_1 \\ \bar{p}_2 \\ \vdots \\ \bar{p}_m \end{bmatrix} \tag{2.5}$$

$$||p_{ij} - \bar{p}_i|| = \sqrt{\sum_{i=1}^{m}(p_{ij} - \bar{p}_i)^2} \tag{2.6}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{m}(p_{ij} - \bar{p}_i)^2} \tag{2.7}$$

Quantifying the spread of the swarm is only useful if an understanding of what the value indicates is developed. The initial value of the standard deviation of the swarm is dictated by the range of uniformly random values used during instantiation of the particle locations. It is possible for the value to rise beyond this initial point during early iterations however it will approach zero as the swarm converges on an optimum.

When the standard deviation reaches zero all particles have occupied the same location in $\Re^m$. This condition is indicative of a complete loss of spatial diversity. Without the ability to compare more than one point on the surface the swarm is ineffective in searching for a more optimal solution.

2.4.3 Distance Moved by Global Best Solution

If the surface being searched is not smooth, spikes in the standard deviation will be observed when the swarm settles into a local optimum temporarily and then breaks out. These sudden rises can also be caused by the random factor in each term providing a large jump in a particles location unrelated to its proximity to its personal best location and the global best location.

A useful tool in discerning the cause of a rise in standard deviation is the distance moved by the global best location from one iteration to the next. If a rise in standard deviation is caused by a move in the global best value the plot of distance will show a large move preceding the rise in standard deviation. These phenomena do not coincide in time perfectly since the swarm will take time to accelerate toward the new global best value.

In the case that the increased standard deviation is cause by the random factor no move by the global best value is observed prior to the spike. While the standard deviation is useful in understanding how the swarm is behaving, the distance plot is useful in understanding the cause behind that behavior. The expression for this metric is shown in equation 2.8 where i and t indicate the dimension and iteration respectively.

$$||g_{it} - g_{it-1}|| = \sqrt{\sum_{i=1}^{m}(g_{it} - g_{it-1})^2} \tag{2.8}$$

2.5 Typical Behavior of Metrics on a Well-behaved Surface

These metrics are not limited to problems of higher dimensions. Observing the behavior of them on a simple surface will benefit understanding the nature of a higher dimensioned surface using the same methods of measure. The two dimensional evaluation function chosen for this purpose is equation 2.9 and is plotted in figure 2.1. For this example the swarm is tasked with finding a maximum value which is known to be located at the origin and have a value of 4. The population of the swarm was 10 and each particle was initialized within ± 100 on each axis.

$$f(x, y) = \left[1 + \frac{cos(x)}{1 + .001x^2}\right]\left[1 + \frac{cos(y)}{1 + .001y^2}\right] \tag{2.9}$$

The progression of the global best value is shown in figure 2.2. The plot shows that while the swarm does initially get caught in a few local maxima, it ultimately finds the global maximum of 4 at the origin. The low population size of the swarm contributed to the length of time that the global best solution spent in the local maxima. A larger swarm performs better on this surface but does not provide a useful demonstration.

Examining the standard deviation of the swarm, as shown in figure 2.3, shows the spread of the swarm as it searches the surface. The plot shows the swarm initially spreading out in the first 50 iterations. Through the rest of the simulation the standard deviation decreases greatly from its maximum value and approaches zero. If the swarm were allowed to search indefinitely the standard deviation would reach zero, a trend

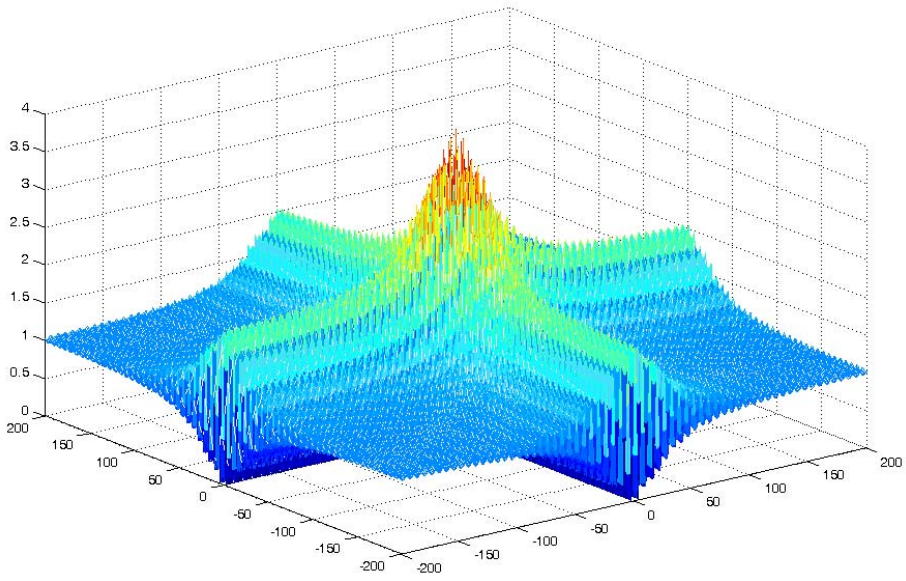

Figure 2.1: Test surface used for metrics demonstration

which is apparent in this figure. The small variation in this value from one iteration to the next can be caused by different factors which leads to the inclusion of the last metric.

The distance moved by the global best solution allows for examination of why the swarm is behaving as it does at various points during the search. Comparing the distance plot with the standard deviation plot shows that the early increase in standard deviation, before the hundredth iteration, can be primarily attributed to the movement of the global best solution. This cause and effect is also seen near the 350TH iteration. The global best plot shows that it is near this iteration that the global optimum is discovered. It is also at this time that a small move is made by the global best solution and a corresponding rise in standard deviation can be observed in the appropriate plot.

The other cause of rising standard deviation is the normal random factors in each term causes a large jump of individual particles. This phenomena can be observed around

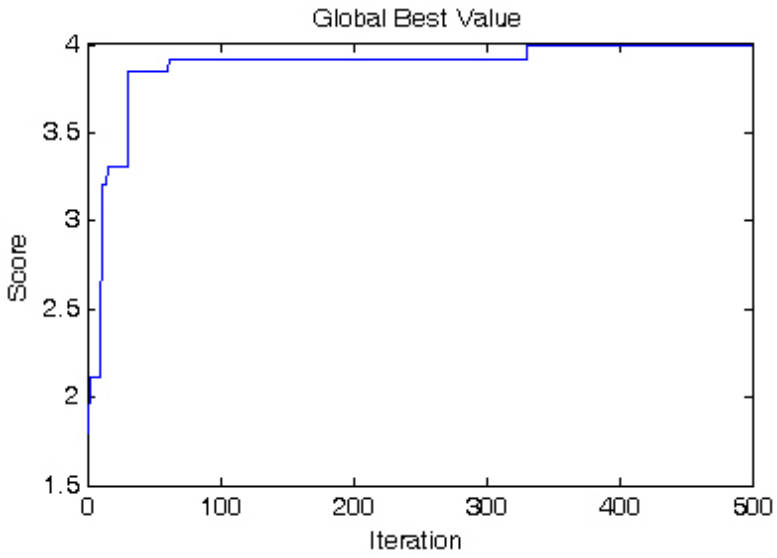

Figure 2.2: Global best value of swarm on test surface

the 225TH iteration. The global best plot shows that no progression in optimal value is made at this time and the distance plot shows that there is no movement of the solution on the surface. However, examining the standard deviation plot at this point in time shows a small rise in standard deviation.

These metrics and an understanding of how they relate to the surface being searched offer a method of evaluating swarm performance over problems of varying dimensions.

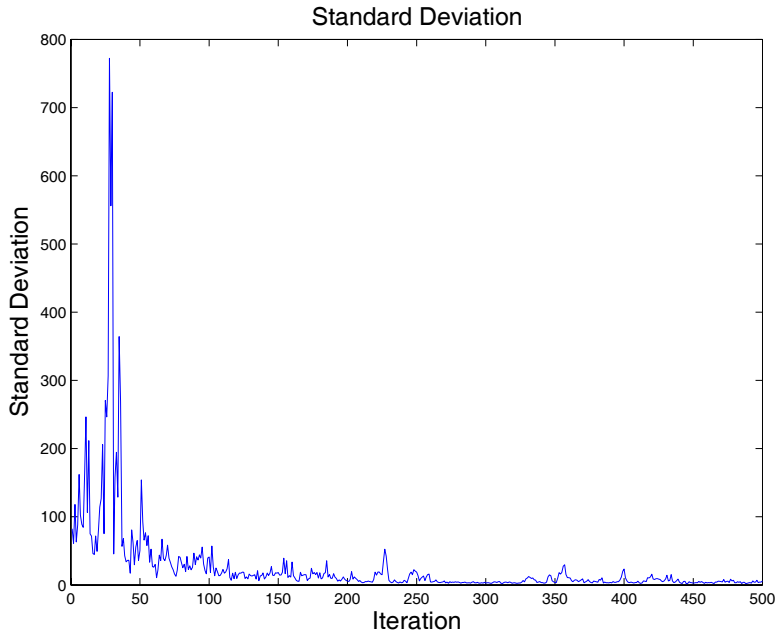

Figure 2.3: Standard deviation of swarm on test surface

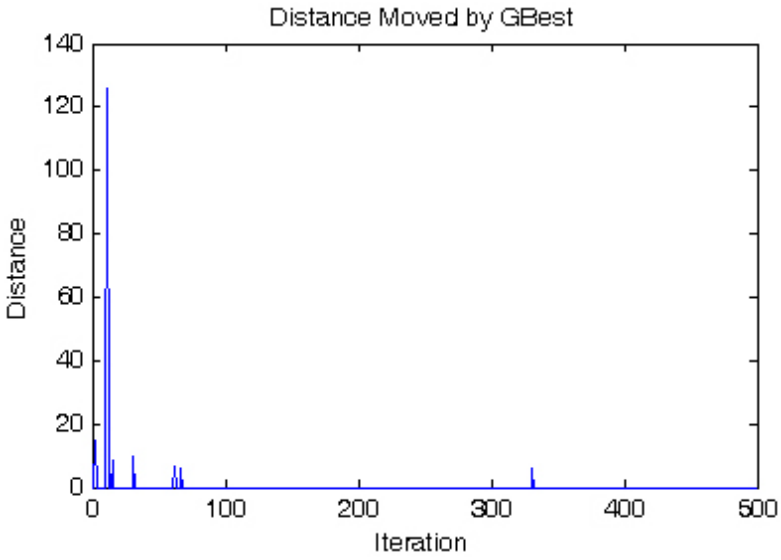

Figure 2.4: Distance moved by global best solution on test surface

Chapter 3

Application to Nonlinear Control

PSO's history of being applied to linear controls and, more specifically, PID control has been previously detailed in section 1.1. Applications of PSO to nonlinear controls have not been explored to the same extent and warrants examination. As a first step in this examination, input to state linearization was chosen as a nonlinear method to which PSO could be applied.

A system with well known dynamics and whose linearizing control is easily derived is considered. This system was chosen to test PSO's ability to search a nonlinear surface in an effort to linearize the plant's dynamics. The parallel plate actuator serves this purpose well and allows for the solution developed by the swarm to be compared to a known value that has been classically designed.

3.1 Plant Model

The states of the second order model are described in equation 3.1 where the coefficients c_1 through c_3 are calculated as in equation 3.2. The variables used in both equations are described in table 3.1. The states, z_1 and z_2, represent the position and velocity of the moving plate respectively.

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= c_1 z_1 + c_2 z_2 + c_3 (h + z_1)^{-2} v^2 \end{aligned} \tag{3.1}$$

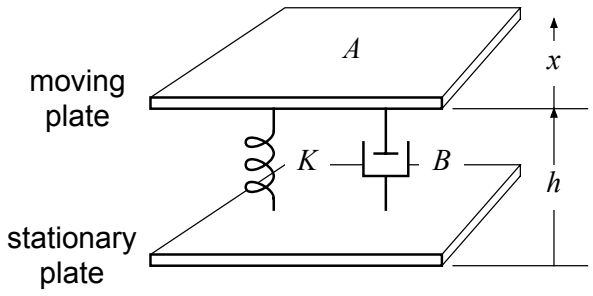

Figure 3.1: Parallel Plate Actuator

Symbol	Description	Value	Units
ϵ	permeability	8.854×10^{-12}	F/m
m	mass of moving plate	1.21025×10^{-4}	kg
K	stiffness constant	3.4×10^{3}	N/m
B	damping constant		
A	area of moving plate	10^{-4}	m^2
x	displacement of moving plate(plant output)		m
h	nominal gap between plates	10^{-5}	m
v	actuator voltage(plant input)		V

Table 3.1: Parallel Plate Actuator Nomenclature

$$c_1 = -\frac{K}{m} \qquad c_2 = -\frac{B}{m} \qquad c_3 = -\frac{\epsilon A}{2m} \tag{3.2}$$

3.2 Review of MEMS Actuator Control Methods

Control of the PPA itself has proven to be a difficult problem in and of itself. Summarized here are previous attempts to provide a control method for the system. Many detail methods of avoiding the nonlinearities inherent in the plant. Only recently have nonlinear methods been explored for controlling the plant.

The force of electrostatic origin is a nonlinear function of input voltage and actuator displacement. A well known problem is the so-called 'pull-in' or 'snap in' effect, where the nonlinear electrostatic force avalanches over the linear spring force, causing the parallel plates to snap together. In the open loop case, the snap in effect limits the useful displacement to one-third of the nominal gap between the plates.

Several methods exist for controlling electrostatic MEMS actuators. A simple, open loop method of increasing the controllable range of motion of a PPA, as described by Bao [7], is to design the actuator to have a rest distance that is three times the required range of motion. In this way, the actuator never leaves the open loop stable operating range. A second but equivalent technique to realize this effect is to put an appropriately sized capacitor in series with the actuator, which makes the capacitor-actuator combination appear electrically as an actuator with a rest gap three times larger than it actually has. A drawback of these techniques is the requirement of a much larger input voltage.

Chen et al. [8] employ a modest linear gain schedule to improve motion in a MEMS optical switch application. Lu and Fedder propose a two-degree-of-freedom control structure, in which a linear compensator and linear prefilter are employed [9]. These methods

have the advantage of being simple to design and implement, but do not capture all of the effects of the actuator nonlinearity. Actuator nonlinearity can lead to significant performance problems, including chaotic oscillations [10].

The extreme nonlinearity of the electrostatic actuator can be partially addressed by the physical layout and design of the actuator. For example, Rosa et al. [11] describe an external, tapered electrode to compensate for the nonlinear relationships between force, voltage, and gap distance. Chiou and Lin [12] use multiple electrodes to cover the operating range by several smaller motions.

Novel control approaches include the use of charge control (Seeger and Boser [13]). Using charge control, the extreme nonlinearity of force with respect to voltage and gap distance is largely avoided. A charge amplifier is required, however, and the approach does have some sensitivity to parasitic capacitance. Seeger and Boser have extended the application of the charge amplifier to a feedback configuration that produces the effect of negative capacitance [14], which also improves dynamic stability.

Chu and Pister [15] simulate a nonlinear control law that employs a square-root characteristic to compensate for the nonlinear relationship of force vs (voltage, distance). That control is straightforward to derive from the 2nd-order mechanical dynamics model and can be expressed in analytical form, but implementation was not easy to perform or test with the technologies of the early 1990s.

More recently, Maithripala et al. simulate a nonlinear control law with a nonlinear state estimator [16]. The state estimator is proposed as a means to circumvent the full state feedback requirement.

In general, nonlinear control design by the above methods can be mathematically tedious, or even intractable.

3.3 Analytical Solution

The model in equation 3.1 was selected to provide a problem with a known solution so that result could be verified. This model can be examined and the control law v can be designed to cancel the nonlinearities in $\dot{z_2}$. The designed v is shown in equation 3.3. Expanding v results in equation 3.4 which allows the coefficients to be parameterized and the control law to be rewritten as in equation 3.5.

$$v = \left[\frac{(h+z_1)^2}{c_3} \left(-c_2 z_2 - c_1 z_1 + r \right) \right]^{\frac{1}{2}} \tag{3.3}$$

$$\begin{aligned} v \quad = \quad & \left[\frac{-c_1 h^2}{c_3} z_1 + \frac{-2c_1 h}{c_3} z_1^2 + \frac{-c_1}{c_3} z_1^3 \right. \\ & + \frac{-c_2 h^2}{c_3} z_2 + \frac{-2c_2 h}{c_3} z_1 z_2 + \frac{-c_2}{c_3} z_1^2 z_2 \\ & \left. + \frac{h^2}{c_3} r + \frac{2h}{c_3} z_1 r + \frac{1}{c_3} z_1^2 r + \right]^{\frac{1}{2}} \end{aligned} \tag{3.4}$$

$$\begin{aligned} v \quad = \quad & \Big[p_{1j} z_1 + p_{2j} z_1^2 + p_{3j} z_1^3 \\ & + p_{4j} z_2 + p_{5j} z_1 z_2 + p_{6j} z_1^2 z_2 \\ & + p_{7j} r + p_{8j} z_1 r + p_{9j} z_1^2 r + \Big]^{\frac{1}{2}} \end{aligned} \tag{3.5}$$

Substituting equation 3.3 into equation 3.1 results in the closed-loop dynamics described in equation 3.6. The resultant system has two poles located at the origin indicating that it is only marginally stable.

$$\begin{aligned} \dot{z}_1 &= z_2 \\ \dot{z}_2 &= r \end{aligned} \tag{3.6}$$

While the control law described in equations 3.3 through 3.5 would offer linear dynamics for the closed loop system it would not provide a stable system. Two methods present themselves to further stabilize the system. Pole placement can be performed using linear techniques and an additional outer loop would be added to the system. The alternative is to alter the control law in a manner which will place the poles directly.

3.4 Selection of Desired Dynamics

The desired dynamics of the system were chosen to be simple yet still offer exponentially stable characteristics. A system with characteristic equation $s^2 + 2s + 1$ has two poles at $s = -1$ therefore meeting both criteria. However, further investigation is necessary to determine whether the system can be reasonably expected to behave in such a manner.

The first method to be examined is the addition of an outer loop using linear techniques to place the poles in the left half-plane. The closed-loop dynamics described in equation 3.6 results in a system that has the transfer function $1/s^2$. The root locus of this system shows the two poles at the origin move to infinity along the imaginary axis as increasing negative feedback is applied. This inidcates that, without further compensation, the system can not meet the chosen desired dynamics.

The second method would be to redesign the control law in equation 3.3 to cause the closed-loop system to meet the desired dynamics. With the current control the state-space representation of the closed-loop system appears as in equation 3.7. The eigenvalues, λ, of the system matrix reveal that the two poles are located at the origin. Redesigning this control may allow for the system to behave as desired.

$$\dot{\mathbf{z}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{z} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r \tag{3.7}$$

The desired dynamics call for the two poles of the system to be located at -1. Equation 3.8 describes the calculation of the eigenvalues of a standard 2x2 matrix of the form in equation 3.9. It is important to note that the open-loop dynamics only allow the input to affect $\dot{z_2}$ directly. Therefore, a_{11} and a_{12} can not be altered from 0 and 1 respectively. Using the two desired pole locations and the two equations in equation 3.8 created by the $\pm$ a system is formed that can be solved simultaneously.

$$\begin{aligned} \lambda &= \frac{1}{2}\left[(a_{11}+a_{22}) \pm \sqrt{4a_{12}a_{21}+(a_{11}-a_{22})^2}\right] \\ &= \frac{1}{2}\left[(0+a_{22}) \pm \sqrt{4(1)a_{21}+(0-a_{22})^2}\right] \\ &= \frac{1}{2}\left[a_{22} \pm \sqrt{4a_{21}+a_{22}^2}\right] \end{aligned} \tag{3.8}$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{3.9}$$

In this case, the resulting system matrix is seen in equation 3.10 where a_{21} and a_{22} are -1 and -2 respectively. These calculations form the constraints placed on the selection of the desired dynamics if an outer-loop is not employed. The system of equations formed regarding the eigenvalues of the system matrix must be consistent in order for the closed-loop system to behave as desired. It is not necessary for it to produce a unique solution since any solution that will place the poles appropriately is acceptable.

$$\dot{\mathbf{z}} = \begin{bmatrix} 0 & 1 \\ -1 & -2 \end{bmatrix} \mathbf{z} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r \tag{3.10}$$

The control law in equation 3.3 needs to be altered to accommodate the desired dynamics in this case. By adding two terms to the expression, as seen in equation 3.11, the closed-loop system matrix will match the desired form. Expanding this redesigned expression results in equation 3.12. Once again, the coefficients can be parameterized as in equation 3.5 since only they have changed and not the form of the terms.

$$v = \left[\frac{(h+z_1)^2}{c_3} \left(-c_2 z_2 - c_1 z_1 + r \quad \underbrace{-z_1 - 2z_2}_{additional\ terms} \right) \right] \tag{3.11}$$

$$\begin{aligned} v \quad = \quad & \left[\frac{(-c_1-1)h^2}{c_3} z_1 + \frac{2(-c_1-1)h}{c_3} z_1^2 + \frac{-c_1-1}{c_3} z_1^3 \right. \\ & + \frac{(-c_2-2)h^2}{c_3} z_2 + \frac{2(-c_2-2)h}{c_3} z_1 z_2 + \frac{-c_2-2}{c_3} z_1^2 z_2 \\ & \left. + \frac{h^2}{c_3} r + \frac{2h}{c_3} z_1 r + \frac{1}{c_3} z_1^2 r + \right]^{\frac{1}{2}} \end{aligned} \tag{3.12}$$

3.5 Implementation Issues

PSO has been implemented and tested frequently on academic problems as well as simple linear control problems. Applying this technique to the design problem posed by the PPA provides its own challenges that are not uncommon to implementing PSO in a practical manner. The choices made surrounding these issues can have consequences on the success, performance, and implications on the significance of the results.

The first of these issues is choosing a meaningful representation of the candidate solutions that the swarm will be able to operate on. While the swarm is capable of searching a surface with discontinuities in some cases it is desirable to limit the frequency of this type of feature. Furthermore, it is necessary for the choice of representations to meet the closure principle. That is, it must provide a search space in which every possible location is a valid, meaningful solution that can then be evaluated.

The swarm's ability to adjust a large number of parameters must be used in relation to the problem at hand. Examining the analytical solution previously discussed offers a useful representation of the solutions. In particular, equation 3.12, and its parameterized representation in equation 3.5, can be employed to translate a particle's location on a nine-dimensional surface into a possible tuning of the linearizing control defined as v.

It is also useful, although not entirely necessary, to have a general idea of the neighborhood in which the optimal solution exists. The particles are uniformly distributed over an initial range which should be targeted toward this neighborhood. This is not to say that if the swarm is initialized in a different area that it will not be able to find the optimum. PSO's performance in this instance depends on other factors including the

nature of the surface being searched. Initializing the swarm outside the optimum's neighborhood will lower the percentage of successful runs in which the optimum is located accurately without getting stuck in a local optimum.

3.6 Evaluation of Solutions

The means used to evaluate each particle location plays a large role in the ability of the swarm to optimize effectively. In an effort to quantify how closely a controller's response matches that of a desired dynamic a model reference approach was taken. Figure 3.2 shows the SIMULINK model used in running the evaluation method. The top branch of the diagram is the expression for the desired dynamics described in previous sections. The bottom branch is the controller and plant model which is used to simulate a particular tuning of the controller according to a particle's location on the surface.

The time period the system simulation is performed over plays a role in determining how likely the system is to converge on the true parameters and how computationally efficient it is. If the simulation is too short the swarm will not have enough information to properly score each candidate solution. In this instance multiple solutions can provide the same evaluation score. While this will not be evident in the swarm metrics it has consequences when examining the systems response over a longer time or to a different reference signal.

If the simulation time is too long the algorithm will be inefficient and time-consuming. The swarm runs the evaluation method for each particle and for each iteration. Any time wasted in evaluating a solution is multiplied over the entire run of the swarm. Therefore, it is important to chose a simulation time that provides a meaningful amount of information without being redundant.

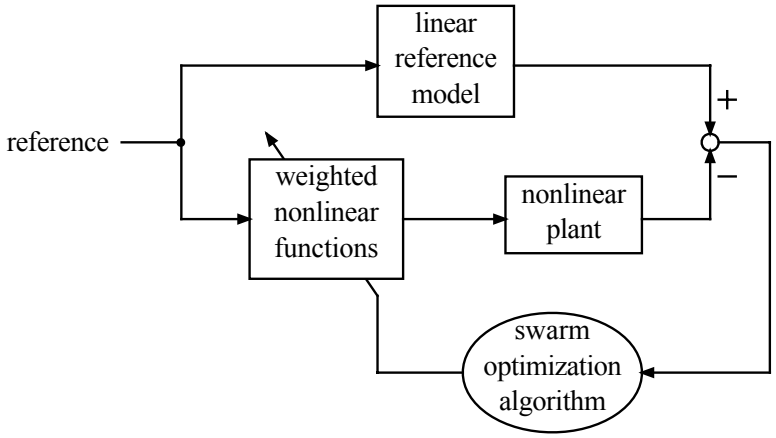

Figure 3.2: Model reference system used in evaluating candidate solutions.

This particular problem poses a unique difficulty. Examination of the analytical solution reveals that the control is ill-defined over the range of operation. The root that encloses the expression causes the controller to produce imaginary results whenever the operand of the function is a negative value. This effectively limits the validity of certain control tunings and even has implications on the validity of the analytically designed control law.

Approaching this problem from a practical perspective any implementation of the controller must be able to handle a tuning of the controller that is capable of producing negative values under the square-root. The SIMULINK controller as seen in figure 3.3 is implemented to limit the values under the root to positive values through the use of the saturation block.

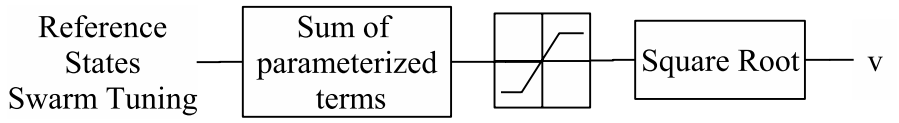

Figure 3.3: Model of linearizing controller using particle location as input.

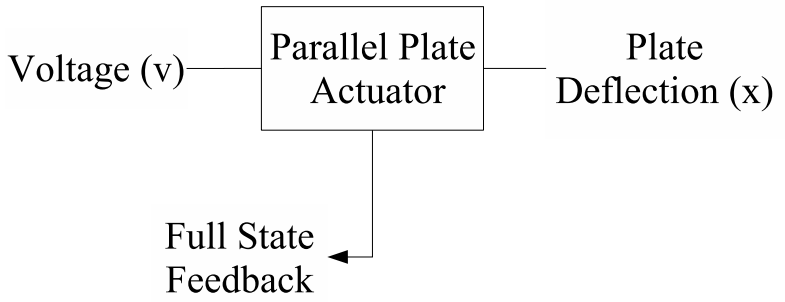

Figure 3.4: Model of the parallel plate actuator used in simulation.

Chapter 4

Resultant Tuning and Performance

Having implemented and tested both the analytically designed control and the control provided by the swarm, the results are presented here.

4.1 Response Comparison

The purpose of choosing the PPA for evaluation of this technique was to compare the analytically designed control to the expression discovered by the swarm. Figure 4.1 shows the response of the plant with the designed controller in comparison to the same model in Figure 4.2. Visual inspection reveals that while the responses are similar, the analytically designed response matches the model more closely.

To quantify this difference the ISE of each response is examined. The PSO designed response yields an ISE of 1.1002e-017 while the analytically designed response yields an ISE of 4.2049e-017. Both responses closely match that of the model's. However, it is notable that the PSO designed control performs better than the analytically designed control.

It must also be noted that the analytical solution does not provide a perfect result. Ideally the ISE of the tuning would equal zero. In both cases, that of the PSO controller and the designed controller, the plant response does not track the model response perfectly. For the analytical control, this can be attributed to the discontinuity introduced by the root function. The PSO control successfully compensates for this difficulty.

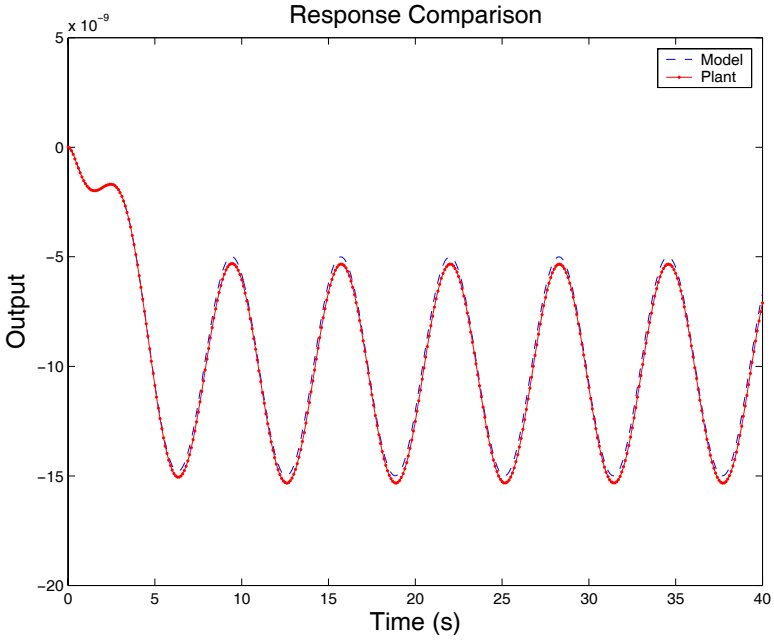

Figure 4.1: Response of system with analytically designed control.

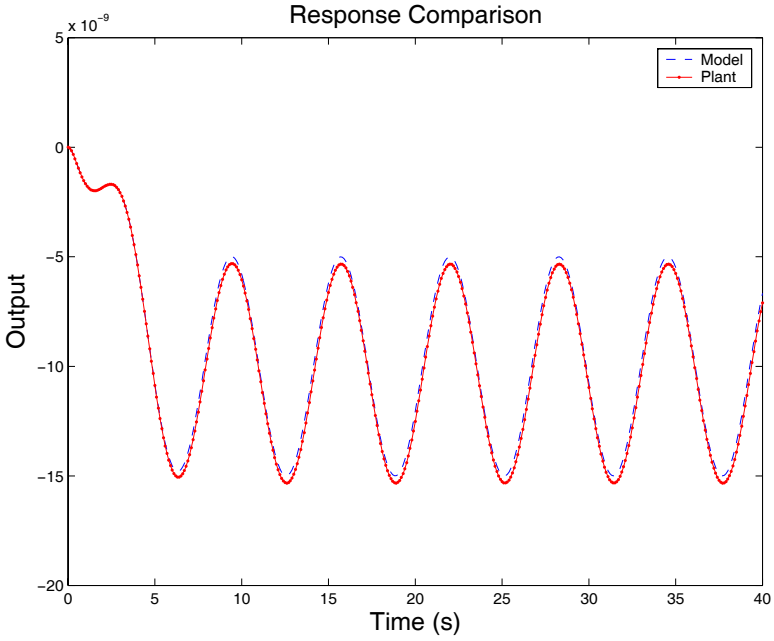

Figure 4.2: Best tuning of the controller found by swarm.

4.2 Behavior of Metrics

The swarm iterated through 300 iterations in 9 dimensions using the model reference evaluation method. In figure 4.3 it is seen that the global best value converged to a point near its final value within the first 50 iterations. While minimal improvement is seen beyond this point in the simulation it is useful to examine the behavior of the other metrics over a longer period of time.

In examining the standard deviation of the swarm over time it is seen that the expected behavior is present. The overall trend, apparent in figure 4.4, of the value approaches zero as time increases. Small perturbations are still present and are more frequent than in the test surface used earlier. This behavior is consistent with the swarm performing a global search early in the simulation and then converging toward an optimal value and performing a local search later in the simulation.

Using the distance plot in conjunction with the standard deviation shows that the initial 50 iterations affected the spread of the swarm. Later spikes in standard deviation can be attributed to the probabilistic factors in the velocity equations causing acceleration in the swarm. Small adjustments in the global best value are seen after the 50TH iteration. In a similar manner as the standard deviation plot this indicates the swarm is searching a larger area with less resolution early in the simulation only to congregate in one smaller area and provide a greater resolution making only small changes to the optimal location.

The behavior of these metrics speak to the highly nonlinear nature of the surface as a more well behaved surface would allow for faster convergence of all of the particles

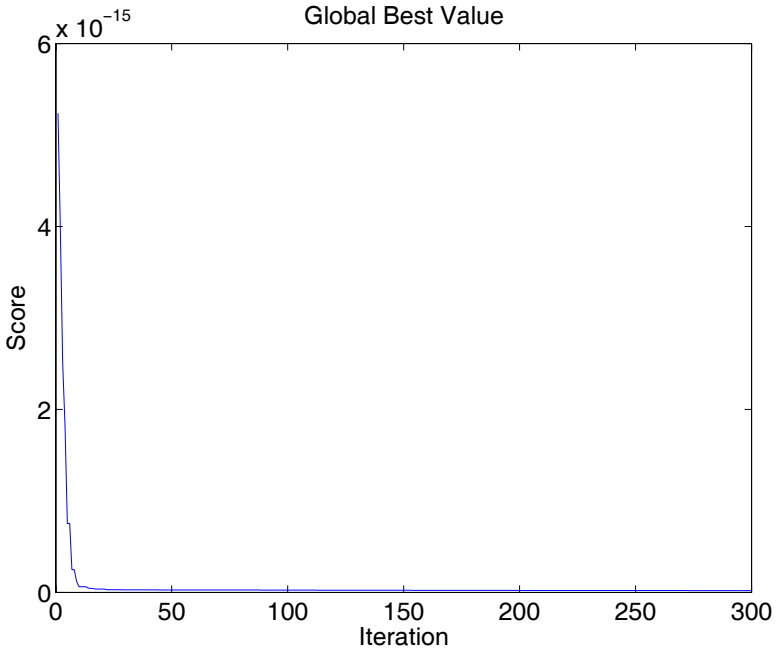

Figure 4.3: Global best value of swarm while tuning linearizing control.

on a single value. In this situation PSO offers greater probability of converging on the global optimum than traditional optimization techniques.

4.3 Parameter Convergence

Given that the designed parameters are known the optimal particle location on the surface is also known. The performance of the swarm can be evaluated using the designed parameters as a benchmark. If a comparison of these values against those found by the swarm shows that the global best solution lies on the designed values than it can be said that the swarm has found the global optimum. If they do not, the global optimum has not been successfully found.

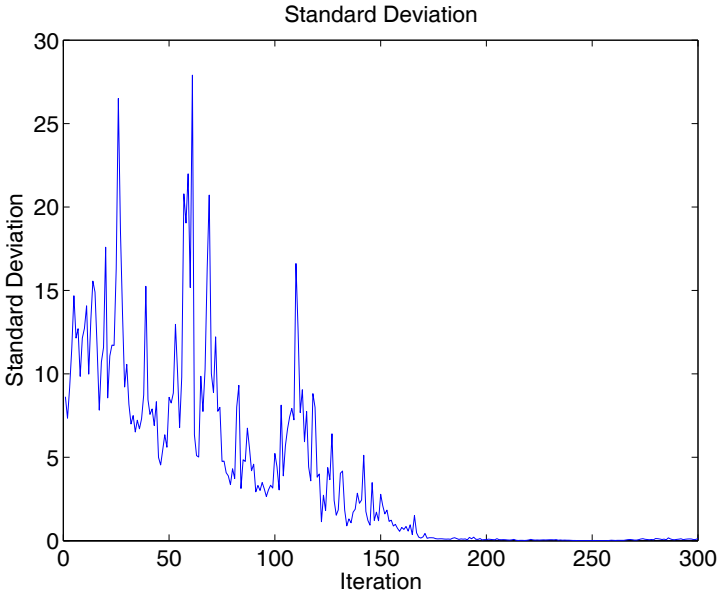

Figure 4.4: Standard deviation of swarm while tuning linearizing control.

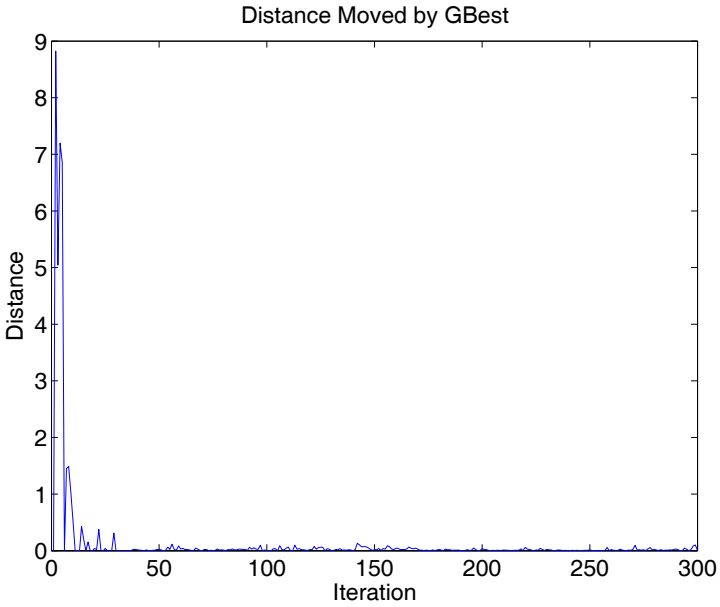

Figure 4.5: Distance moved by global best solution while tuning linearizing control.

Parameter	Analytical	Sine
p_{1j}	-2.7338	-3.1994
p_{2j}	-7.4068	-7.1316
p_{3j}	5.2417	4.0201
p_{4j}	-0.0741	0.3250
p_{5j}	-0.0027	1.7561
p_{6j}	0.0052	-5.6807
p_{7j}	-7.4068	-34.7615
p_{8j}	-2.7338	-7.1671
p_{9j}	5.2417	-0.2572

Table 4.1: Parameter comparison

The results for the application of PSO to the PPA are shown in Table 4.1 where the parameters are listed as they appear in equation 3.5. It is seen that the swarm failed to converge on the parameters as designed through analytical methods. This does no negate any success the swarm had in matching the response of the plant/controller to that of the model. Rather, it speaks to the nature of the surface and indicates that it is, in fact, nonlinear and rife with local optima. This serves to bolster the argument for PSO as opposed to traditional optimization techniques. While PSO did not find the known solution it did find a solution with a similar, albeit larger, fitness score. Both solutions offer a dynamic response that closely match that of the model's.

4.4 Model Limitations

The reference signal used to excite the model system must be chosen with consideration to the nature of the plant being controlled. The PPA plant represents the deflection of the moving plate from a nominal position as the variable to control. A control value does not exist that can repel the plates from each other, only attract them together.

Taking this into consideration leads to the conclusion that the system can only relax at it's natural rate caused by the open loop dynamics of the system.

This realization limits the desired response in two ways. The first requires that the desired dynamics and the reference signal do not call for the moving plate to travel above its nominal position, h. The second requires that any movement of the moving plate in the positive direction occur at a rate which the open loop system can accommodate.

Figure 4.6 shows the result of an attempt to tune the swarm to an inappropriate reference signal. This example removes the offset value in the input to cause the desired system response to enter the positive region. Examining the best tuning the swarm was able to find illustrates the limitations of this model and displays PSO's ability to find a reasonable response despite unreasonable expectations. It also demonstrates that while PSO is a powerful tool for tuning the controller it does not replace the designer's knowledge and understanding of the system.

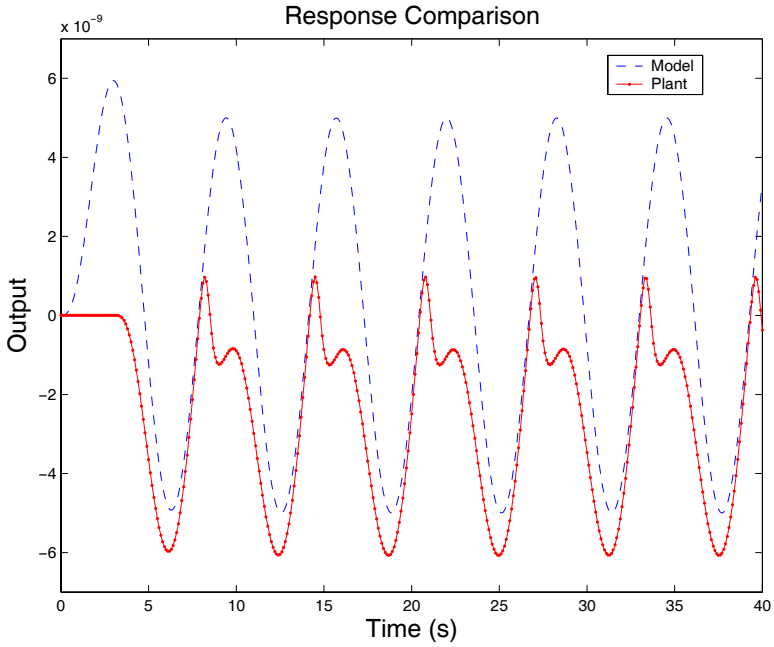

Figure 4.6: Best tuning of the controller with an inappropriate reference signal.

Chapter 5

Conclusions and Future Work

5.1 The Good

The use of PSO as a means for tuning an input to state linearizing controller has proved itself as a viable method. While the results did not meet those of the analytical method, it was the intention of this work to show that the technique could be used to achieve acceptable results. Further investigation should examine the role of the reference signal used to excite the model and plant for evaluation. A richer signal may provide better results and a more suitable response.

5.2 The Bad

The control used in this example makes the assumption that the parameters appear linearly in the control. In examining the case in which the control is nonlinear by parameters it was found that the continuous implementation of PSO was not suitable. As the swarm adjusted the parameters that appeared as exponents, any fractional value in the parameter appeared as a root function in the candidate solution. This effectively limited the control to positive values and caused the majority of solutions to be invalid. In this case, the requirement to meet the closure principle was not met and the swarm could not effectively search.

It may be possible to use the discrete implementation of PSO also developed by Kennedy and Eberhart to overcome this difficulty. A discrete representation of the

solution would segment the parameters in different manners depending on where they appear in the control.

5.3 The Ugly

While this technique of tuning the control law is immediately useful, further study should be directed in an effort to determine the structure of the control law in addition to the tuning. The combination of these two capabilities would prove invaluable.

Using PSO to perform this search may prove difficult. In both the continuous and discrete implementations of PSO the surface being searched must be predominantly continuous. In varying a control structure from one function to the next the surface would not meet this requirement. John Koza's work in Genetic Programming offers a more readily available solution to this task.

Overall, this technique of applying PSO to a nonlinear control problem shows promise. Setting up the design engine proved simple while still yielding acceptable results.

Bibliography

[1] Y. L. J. Z. S. Wang, "Optimization design based on pso algorithm for pid controller," *Fifth World Congress on Intelligent Control and Automation,*, vol. 3, pp. 2419–22, June 2004.

[2] Z.-L. Gaing, "A particle swarm optimization approach for optimum design of pid controller in avr system," *IEEE Transactions on Energy Conversion*, vol. 19, no. 2, pp. 384–91, June 2004.

[3] H. H. Q. H. Z. L. D. Xu, "Optimal pid controller design in pmsm servo system via particle swarm optimization," *32nd Annual Conference of IEEE Industrial Electronics Society*, pp. 79–83, November 2005.

[4] L.-y. Z. J.-x. Q. Yong-ling Zheng, Long-hua Ma, "Robust pid controller design using particle swarm optimizer," *IEEE International Symposium on Intelligent Control*, pp. 974–9, October 2003.

[5] R. Kennedy, J. Eberhart, "Particle swarm optimization," *IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, Nov. 1995.

[6] R. Krohling, "Gaussian swarm: a novel particle swarm optimization algorithm," *IEEE Conference on Cybernetics and Intelligent Systems*, vol. 1, pp. 372–376, Dec. 2004.

[7] M.-H. Bao, *Handbook of Sensors and Actuators*. New York: Elsevier Science, 2000, vol. 8.

[8] J. Chen, W. Weingartner, A. Azarov, and R. C. Giles, "Tilt-angle stabilization of electrostatically actuated micromechanical mirrors beyond the pull-in point," *Journal of Microelectromechanical Systems*, vol. 13, no. 6, pp. 988–997, December 2004.

[9] M. S.-C. Lu and G. K. Fedder, "Position control of parallel plate microactuators for probe-based data storage," *Journal of Microelectromechanical Systems*, vol. 13, no. 5, pp. 759–768, October 2004.

[10] S. Liu, A. Davidson, and Q. Lin, "Simulation studies on nonlinear dynamics and chaos in a MEMS cantilever control system," *Journal of Micromechanics and Microengineering*, vol. 14, pp. 1064–1073, June 2004.

[11] M. A. Rosa, D. D. Bruyker, A. R. Völkel, E. Peters, and J. Dunec, "A novel external electrode configuration for the electrostatic actuation of MEMS based devices," *Journal of Micromechanics and Microengineering*, vol. 14, pp. 446–451, January 2004.

[12] J.-C. Chiou and Y.-C. Lin, "A multiple electrostatic electrodes torsion micromirror device with linear stepping angle effect," *Journal of Microelectromechanical Systems*, vol. 12, no. 6, pp. 913–920, December 2003.

[13] J. I. Seeger and B. E. Boser, "Charge control of parallel-plate, electrostatic actuator and the tip-in instability," *Journal of Microelectromechanical Systems*, vol. 12, no. 5, pp. 656–671, October 2003.

[14] ——, "Negative capacitance for control of gap-closing electrostatic actuators," in *12th International Conference on Solid-State Sensors, Actuators and Microsystems*, Boston, MA, June 2003, pp. 484–487.

[15] P. B. Chu and K. S. J. Pister, "Analysis of closed-loop control of parallel-plate electrostatic microgrippers," in *Proceedings of 1994 International Conference on Robotics and Automation*, San Diego, CA, May 1994, pp. 820–825.

[16] D. H. S. Maithripala, R. O. Gale, M. W. Holtz, J. M. Berg, and W. P. Dayawansa, "Nano-precision control of micromirrors using output feedback," in *Proceedings of 42nd IEEE Conference on Decision and Control*, vol. 3, December 2003, pp. 2652–2657.

Appendices

Appendix A

Code Listing used in Simulations

A.1 Plotting Code for Swarm Test Surface

test_mesh.m was used to prepare figure 2.1.

```
clear all
close all
clc

[x,y]=meshgrid(-200:1:200,-200:1:200);
out=(1+(cos(x)./(1+.001.*(x).^2))).*(1+(cos(y)./(1+.001.*(y).^2)));
mesh(x,y,out);
```

A.2 Implementation of Gaussian Particle Swarm for Testing

param_ISLGPSO.m was used to generate figures 2.2, 2.3, and 2.4.

```
%Evaluation of Swarm Parameters

%Initialization
clear all
close all
clc
warning off all
randn('state',[143;06084]);
rand('state',9121914);

%SETUP VARIABLES
max_ite=500;            %maximum number of iterations
init_range=200;          %initial location range
m=2;                    %dimensions
n=10;                   %population
```

```
inertial_weight=0;      %velocity weights

t=1;                                %iteration counter
location(1:m,1:n,1:max_ite)=0;      %location of particles
velocity(1:m,1:n,1:max_ite)=0;      %velocity of particles
pbest_loc(1:m,1:n,1:max_ite)=0;     %best location of individual particle
pbest_val(1,1:n,1:max_ite)=0;       %best value of individual particle
gbest_loc(1:m,1,1:max_ite)=0;       %best location of entire swarm
gbest_val(1,1,1:max_ite)=0;         %best value of entire swarm

%inititalize population
location(:,:,t)=init_range*rand(m,n)-(init_range/2);
stddev(t)=psosd(location(:,:,t))
dist(1)=0;
pbest_loc(1:m,1:n,t)=location(1:m,1:n,t);
for x=1:n
    particle=location(:,x,t)';
    pbest_val(1,x,t)=PSO_eval(location(:,x,t));
end
[v,i]=max(pbest_val(1,:,t))
gbest_val(1,1,t)=v;
gbest_loc(:,1,t)=pbest_loc(:,i,t);

%Fly Swarm
for t=2:max_ite
    velocity(:,:,t-1)=inertial_weight.*velocity(:,:,t-1)
     _+(abs(randn(m,n)).*(pbest_loc(:,:,t-1)-location(:,:,t-1)))
     _+(abs(randn(m,n)).*(repmat(gbest_loc(:,:,t-1),[1 n])
     _-location(:,:,t-1)));
    location(:,:,t)=location(:,:,t-1)+velocity(:,:,t-1);
    stddev(t)=psosd(location(:,:,t));
    gbest_val(1,1,t)=gbest_val(1,1,t-1);
    gbest_loc(:,1,t)=gbest_loc(:,1,t-1);
    for x=1:n
        particle=location(:,x,t)';
        v=PSO_eval(location(:,x,t));
        if (v<pbest_val(1,x,t-1))
            pbest_val(1,x,t)=v;
            pbest_loc(:,x,t)=location(:,x,t);
        else
            pbest_val(1,x,t)=pbest_val(1,x,t-1);
            pbest_loc(:,x,t)=pbest_loc(:,x,t-1);
```

```
        end
        if (v>gbest_val(1,1,t))
            gbest_val(1,1,t)=v;
            gbest_loc(:,1,t)=location(:,x,t);
            [t stddev(t)]
            v

        end
    end
    dist(t)=(sum((gbest_loc(:,1,t)-gbest_loc(:,1,t-1)).^2)).^.5;
end

figure(1);
subplot(2,2,1);
plot(shiftdim(gbest_val));
title('Global Best Value');
xlabel('Iteration');
ylabel('Score');

subplot(2,2,2);
plot(stddev);
title('Standard Deviation');
xlabel('Iteration');
ylabel('Standard Deviation');

subplot(2,2,3);
plot(dist);
title('Distance Moved by GBest');
xlabel('Iteration');
ylabel('Distance');
```

A.3 Evaluation Function for Swarm Testing

PSO_eval.m was used to implement the evaluation function shown in figure 2.1.

```
function out=PSO_eval(pos)
out=(1+(cos(pos(1))/(1+.001*(pos(1))^2)))
_*(1+(cos(pos(2))/(1+.001*(pos(2))^2)));
```

A.4 Implementation of Gaussian Particle Swarm for Parallel Plate Actuator

PPA_ISLGPSO.m was used to generate figures 2.2, 2.3, and 2.4.

```
%Input to State Linearization by Particle Swarm Optimization

%Initialization
clear all
close all
clc
warning off all
randn('state',[9141940;6131916]);
rand('state',1121921);

%SETUP MODEL PARAMETERS
K=3.4;
mass=.121025;
B=0.01;
e0=8.854;
A=10^-4;
h=10^-5;
c1=-K/mass;
c2=-B/mass;
c3=(-e0*A)/(2*mass);

design_particle=[((10^8*h^2)/c3) ((10^7*h^2*(-c1-1))/c3)
_((10^8*h^2*(-c2-2))/c3)  ((h*(-c1-1))/c3) (h/c3)
_((h*(-c2-2))/c3) ((-c1-1)/(1000*c3))  (1/(c3*100)) ((-c2-2)/(c3*100))];

%SETUP VARIABLES
max_ite=150;            %maximum number of iterations
init_range=10;          %initial location range
m=9;                    %dimensions
n=50;                   %population
inertial_weight=2;      %velocity weights

t=1;                              %iteration counter
location(1:m,1:n,1:max_ite)=0;    %location of particles
velocity(1:m,1:n,1:max_ite)=0;    %velocity of particles
pbest_loc(1:m,1:n,1:max_ite)=0;   %best location of individual particle
```

```
pbest_val(1,1:n,1:max_ite)=0;     %best value of individual particle
gbest_loc(1:m,1,1:max_ite)=0;     %best location of entire swarm
gbest_val(1,1,1:max_ite)=0;       %best value of entire swarm

%inititalize population
location(:,:,t)=init_range*rand(m,n)-(init_range/2);
stddev(t)=psosd(location(:,:,t))
dist(1)=0;
pbest_loc(1:m,1:n,t)=location(1:m,1:n,t);
for x=1:n
    particle=location(:,x,t)';
    pbest_val(1,x,t)=PSO_eval(location(:,x,t));
end
[v,i]=min(pbest_val(1,:,t))
gbest_val(1,1,t)=v;
gbest_loc(:,1,t)=pbest_loc(:,i,t);

%Fly Swarm
for t=2:max_ite
    velocity(:,:,t-1)=inertial_weight.*velocity(:,:,t-1)
    _+(abs(randn(m,n)).*(pbest_loc(:,:,t-1)-location(:,:,t-1)))
    _+(abs(randn(m,n)).*(repmat(gbest_loc(:,:,t-1),[1 n])
    _-location(:,:,t-1)));
    location(:,:,t)=location(:,:,t-1)+velocity(:,:,t-1);
    stddev(t)=psosd(location(:,:,t));
    gbest_val(1,1,t)=gbest_val(1,1,t-1);
    gbest_loc(:,1,t)=gbest_loc(:,1,t-1);
    for x=1:n
        particle=location(:,x,t)';
        v=PSO_eval(location(:,x,t));
        if (v<pbest_val(1,x,t-1))
            pbest_val(1,x,t)=v;
            pbest_loc(:,x,t)=location(:,x,t);
        else
            pbest_val(1,x,t)=pbest_val(1,x,t-1);
            pbest_loc(:,x,t)=pbest_loc(:,x,t-1);
        end
        if (v<gbest_val(1,1,t))
            gbest_val(1,1,t)=v;
            gbest_loc(:,1,t)=location(:,x,t);
            [t stddev(t)]
            v
```

```
        end
    end
    dist(t)=(sum((gbest_loc(:,1,t)-gbest_loc(:,1,t-1)).^2)).^.5;
end

figure(1);
subplot(2,2,1);
plot(shiftdim(gbest_val));
title('Global Best Value');
xlabel('Iteration');
ylabel('Score');

%ADDED FOR MODEL MATCHING
particle=gbest_loc(:,:,max_ite)'
try
    sim('PSO_test2');
end
subplot(2,2,2);
plot(tout,shiftdim(model),'--b')
hold on
plot(tout,plant,'.-r');
xlabel('Time (s)');
ylabel('Output');
title('Response Comparison');
legend('Model','Plant',4);

subplot(2,2,3);
plot(stddev);
title('Standard Deviation');
xlabel('Iteration');
ylabel('Standard Deviation');

subplot(2,2,4);
plot(dist);
title('Distance Moved by GBest');
xlabel('Iteration');
ylabel('Distance');
```

A.5 Evaluation function used for Parallel Plate Actuator Problem

PSO_eval.m was used to implement the evaluation function.

```
function out=PSO_eval(pos)
try
    sim('PSO_test2');
catch
    er=3e49;
end
out=sum(er.*er);
```

A.6 Model reference SIMULINK model file

PSO_eval.mdl was used to implement the model reference structure for evaluation of solutions.

```
Model {
  Name   "pso_test2"
  Version   5.0
  SaveDefaultBlockParams  on
  SampleTimeColors   off
  LibraryLinkDisplay   "none"
  WideLines   off
  ShowLineDimensions   off
  ShowPortDataTypes   off
  ShowLoopsOnError   on
  IgnoreBidirectionalLines off
  ShowStorageClass   off
  ExecutionOrder   off
  RecordCoverage   off
  CovPath   "/"
  CovSaveName   "covdata"
  CovMetricSettings   "dw"
  CovNameIncrementing   off
  CovHtmlReporting   on
```

```
covSaveCumulativeToWorkspaceVar on
CovSaveSingleToWorkspaceVar on
CovCumulativeVarName   "covCumulativeData"
CovCumulativeReport   off
DataTypeOverride   "UseLocalSettings"
MinMaxOverflowLogging   "UseLocalSettings"
MinMaxOverflowArchiveMode "Overwrite"
BlockNameDataTip   off
BlockParametersDataTip  off
BlockDescriptionStringDataTip off
ToolBar   on
StatusBar   on
BrowserShowLibraryLinks off
BrowserLookUnderMasks   off
Created   "Thu Oct 20 00:27:01 2005"
UpdateHistory   "UpdateHistoryNever"
ModifiedByFormat   "%<Auto>"
LastModifiedBy   "Administrator"
ModifiedDateFormat   "%<Auto>"
LastModifiedDate   "Fri Dec 09 19:54:57 2005"
ModelVersionFormat   "1.%<AutoIncrement:422>"
ConfigurationManager   "None"
SimParamPage   "Solver"
LinearizationMsg   "none"
Profile   off
ParamWorkspaceSource   "MATLABWorkspace"
AccelSystemTargetFile   "accel.tlc"
AccelTemplateMakefile   "accel_default_tmf"
AccelMakeCommand   "make_rtw"
TryForcingSFcnDF   off
ExtModeMexFile   "ext_comm"
ExtModeBatchMode   off
ExtModeTrigType   "manual"
ExtModeTrigMode   "normal"
ExtModeTrigPort   "1"
ExtModeTrigElement   "any"
ExtModeTrigDuration   1000
ExtModeTrigHoldOff   0
ExtModeTrigDelay   0
ExtModeTrigDirection   "rising"
ExtModeTrigLevel   0
ExtModeArchiveMode   "off"
```

```
ExtModeAutoIncOneShot   off
ExtModeIncDirWhenArm   off
ExtModeAddSuffixToVar   off
ExtModeWriteAllDataToWs off
ExtModeArmWhenConnect   on
ExtModeSkipDownloadWhenConnect off
ExtModeLogAll   on
ExtModeAutoUpdateStatusClock on
BufferReuse   on
RTWExpressionDepthLimit 5
SimulationMode   "normal"
Solver   "ode5"
SolverMode   "Auto"
StartTime   "0.0"
StopTime   "40"
MaxOrder   5
MaxStep   "auto"
MinStep   "auto"
MaxNumMinSteps   "-1"
InitialStep   "auto"
FixedStep   "auto"
RelTol   "1e-3"
AbsTol   "auto"
OutputOption   "RefineOutputTimes"
OutputTimes   "[]"
Refine   "1"
LoadExternalInput   off
ExternalInput   "[t, u]"
LoadInitialState   off
InitialState   "xInitial"
SaveTime   on
TimeSaveName   "tout"
SaveState   off
StateSaveName   "xout"
SaveOutput   on
OutputSaveName   "yout"
SaveFinalState   off
FinalStateName   "xFinal"
SaveFormat   "Array"
Decimation   "1"
LimitDataPoints   on
MaxDataPoints   "1000"
```

```
SignalLoggingName   "sigsOut"
ConsistencyChecking   "none"
ArrayBoundsChecking   "none"
AlgebraicLoopMsg   "none"
BlockPriorityViolationMsg "warning"
MinStepSizeMsg   "warning"
InheritedTsInSrcMsg   "warning"
DiscreteInheritContinuousMsg "warning"
MultiTaskRateTransMsg   "error"
SingleTaskRateTransMsg  "none"
CheckForMatrixSingularity "none"
IntegerOverflowMsg   "warning"
Int32ToFloatConvMsg   "warning"
ParameterDowncastMsg   "error"
ParameterOverflowMsg   "error"
ParameterPrecisionLossMsg "warning"
UnderSpecifiedDataTypeMsg "none"
UnnecessaryDatatypeConvMsg "none"
VectorMatrixConversionMsg "none"
InvalidFcnCallConnMsg   "error"
SignalLabelMismatchMsg  "none"
UnconnectedInputMsg   "warning"
UnconnectedOutputMsg   "warning"
UnconnectedLineMsg   "warning"
SfunCompatibilityCheckMsg "none"
RTWInlineParameters   off
BlockReductionOpt   on
BooleanDataType   on
ConditionallyExecuteInputs on
ParameterPooling   on
OptimizeBlockIOStorage  on
ZeroCross   on
AssertionControl   "UseLocalSettings"
ProdHWDeviceType   "ASIC/FPGA"
ProdHWWordLengths   "8,16,32,32"
RTWSystemTargetFile   "grt.tlc"
RTWTemplateMakefile   "grt_default_tmf"
RTWMakeCommand   "make_rtw"
RTWGenerateCodeOnly   off
RTWRetainRTWFile   off
TLCProfiler   off
TLCDebug   off
```

```
TLCCoverage   off
TLCAssertion   off
BlockDefaults {
  Orientation     "right"
  ForegroundColor     "black"
  BackgroundColor     "white"
  DropShadow     off
  NamePlacement     "normal"
  FontName     "Helvetica"
  FontSize     10
  FontWeight     "normal"
  FontAngle     "normal"
  ShowName     on
}
BlockParameterDefaults {
  Block {
    BlockType       Constant
    Value       "1"
    VectorParams1D       on
    ShowAdditionalParam     off
    OutDataTypeMode       "Inherit from 'Constant value'"
    OutDataType       "sfix(16)"
    ConRadixGroup       "Use specified scaling"
    OutScaling       "2^0"
  }
  Block {
    BlockType       Fcn
    Expr       "sin(u[1])"
  }
  Block {
    BlockType       Inport
    Port       "1"
    PortDimensions       "-1"
    SampleTime       "-1"
    ShowAdditionalParam     off
    LatchInput       off
    DataType       "auto"
    OutDataType       "sfix(16)"
    OutScaling       "2^0"
    SignalType       "auto"
    SamplingMode       "auto"
    Interpolate       on
```

```
}
Block {
  BlockType       Integrator
  ExternalReset       "none"
  InitialConditionSource  "internal"
  InitialCondition       "0"
  LimitOutput       off
  UpperSaturationLimit    "inf"
  LowerSaturationLimit    "-inf"
  ShowSaturationPort      off
  ShowStatePort       off
  AbsoluteTolerance       "auto"
  ZeroCross       on
}
Block {
  BlockType       Math
  Operator       "exp"
  OutputSignalType       "auto"
}
Block {
  BlockType       Mux
  Inputs       "4"
  DisplayOption       "none"
}
Block {
  BlockType       Outport
  Port       "1"
  OutputWhenDisabled      "held"
  InitialOutput       "[]"
}
Block {
  BlockType       Saturate
  UpperLimit       "0.5"
  LowerLimit       "-0.5"
  LinearizeAsGain       on
  ZeroCross       on
}
Block {
  BlockType       Scope
  Floating       off
  ModelBased       off
  TickLabels       "OneTimeTick"
```

```
  ZoomMode       "on"
  Grid       "on"
  TimeRange       "auto"
  YMin       "-5"
  YMax       "5"
  SaveToWorkspace       off
  SaveName       "ScopeData"
  LimitDataPoints       on
  MaxDataPoints       "5000"
  Decimation       "1"
  SampleInput       off
  SampleTime       "0"
}
Block {
  BlockType       Sin
  SineType       "Time based"
  Amplitude       "1"
  Bias       "0"
  Frequency       "1"
  Phase       "0"
  Samples       "10"
  Offset       "0"
  SampleTime       "-1"
  VectorParams1D       on
}
Block {
  BlockType       SubSystem
  ShowPortLabels       on
  Permissions       "ReadWrite"
  RTWSystemCode       "Auto"
  RTWFcnNameOpts       "Auto"
  RTWFileNameOpts       "Auto"
  SimViewingDevice       off
  DataTypeOverride       "UseLocalSettings"
  MinMaxOverflowLogging   "UseLocalSettings"
}
Block {
  BlockType       Sum
  IconShape       "rectangular"
  Inputs       "++"
  ShowAdditionalParam     off
  InputSameDT       on
```

```
    OutDataTypeMode       "Same as first input"
    OutDataType       "sfix(16)"
    OutScaling       "2^0"
    LockScale       off
    RndMeth       "Floor"
    SaturateOnIntegerOverflow on
  }
  Block {
    BlockType       ToWorkspace
    VariableName       "simulink_output"
    MaxDataPoints       "1000"
    Decimation       "1"
    SampleTime       "0"
  }
  Block {
    BlockType       TransferFcn
    Numerator       "[1]"
    Denominator       "[1 2 1]"
    AbsoluteTolerance       "auto"
    Realization       "auto"
  }
}
AnnotationDefaults {
  HorizontalAlignment     "center"
  VerticalAlignment     "middle"
  ForegroundColor     "black"
  BackgroundColor     "white"
  DropShadow     off
  FontName     "Helvetica"
  FontSize     10
  FontWeight     "normal"
  FontAngle     "normal"
}
LineDefaults {
  FontName     "Helvetica"
  FontSize     9
  FontWeight     "normal"
  FontAngle     "normal"
}
System {
  Name     "pso_test2"
  Location     [2, 74, 1022, 699]
```

```
    Open     on
    ModelBrowserVisibility  off
    ModelBrowserWidth     212
    ScreenColor     "white"
    PaperOrientation     "rotated"
    PaperPositionMode     "auto"
    PaperType     "usletter"
    PaperUnits     "inches"
    ZoomFactor     "125"
    ReportName     "simulink-default.rpt"
    Block {
      BlockType       ToWorkspace
      Name       " "
      Position       [645, 110, 705, 140]
      VariableName       "model"
      MaxDataPoints       "inf"
      SampleTime       "-1"
      SaveFormat       "Array"
    }
    Block {
      BlockType       SubSystem
      Name       "Control"
      Ports       [3, 1]
      Position       [310, 194, 410, 236]
      TreatAsAtomicUnit       off
      System {
Name "Control"
Location [189, 74, 1199, 689]
Open off
ModelBrowserVisibility off
ModelBrowserWidth 200
ScreenColor "white"
PaperOrientation "landscape"
PaperPositionMode "auto"
PaperType "usletter"
PaperUnits "inches"
ZoomFactor "100"
Block {
  BlockType   Inport
  Name   "r"
  Position   [25, 53, 55, 67]
}
```

```
Block {
  BlockType   Inport
  Name   "PSO"
  Position   [25, 83, 55, 97]
  Port   "2"
}
Block {
  BlockType   Inport
  Name   "x_bar"
  Position   [25, 23, 55, 37]
  Port   "3"
}
Block {
  BlockType   Fcn
  Name   "ISL"
  Position   [195, 42, 235, 78]
  Expr   "((.00000001*u[4]*u[3])+(.0000001*u[5]*u[1])"
"+(.00000001*u[6]*u[2])+(u[7]*u[1]^2)
_+(u[8]*u[1]*u[3])+(u[9]*u[1]*u[2])+(1000*"
"u[10]*u[1]^3)+(100*u[11]*u[3]*u[1]^2)+(100*u[12]*u[2]*u[1]^2))"
}
Block {
  BlockType   Math
  Name   "Math\nFunction"
  Ports   [1, 1]
  Position   [365, 45, 395, 75]
  NamePlacement   "alternate"
  Operator   "sqrt"
}
Block {
  BlockType   Mux
  Name   "Mux"
  Ports   [3, 1]
  Position   [140, 41, 145, 79]
  ShowName   off
  Inputs   "3"
  DisplayOption   "bar"
}
Block {
  BlockType   Saturate
  Name   "Saturation"
  Position   [305, 45, 335, 75]
```

```
  UpperLimit   "1e99"
  LowerLimit   "0"
}
Block {
  BlockType   Scope
  Name   "after sat"
  Ports   [1]
  Position   [390, 114, 420, 146]
  Location   [6, 54, 1020, 659]
  Open   off
  NumInputPorts   "1"
  List {
    ListType     AxesTitles
    axes1     "%<SignalLabel>"
  }
  List {
    ListType     SelectedSignals
    axes1     ""
  }
  YMin   "-1"
  YMax   "1"
  SaveName   "ScopeData6"
  DataFormat   "StructureWithTime"
}
Block {
  BlockType   Scope
  Name   "before sat"
  Ports   [1]
  Position   [295, 114, 325, 146]
  Location   [5, 53, 1029, 771]
  Open   off
  NumInputPorts   "1"
  List {
    ListType     AxesTitles
    axes1     "%<SignalLabel>"
  }
  List {
    ListType     SelectedSignals
    axes1     ""
  }
  YMin   "-1"
  YMax   "1"
```

```
  SaveName   "ScopeData5"
  DataFormat   "StructureWithTime"
}
Block {
  BlockType   Outport
  Name   "v"
  Position   [430, 53, 460, 67]
}
Line {
  SrcBlock   "x_bar"
  SrcPort   1
  Points   [30, 0; 0, 20]
  DstBlock   "Mux"
  DstPort   1
}
Line {
  SrcBlock   "r"
  SrcPort   1
  DstBlock   "Mux"
  DstPort   2
}
Line {
  SrcBlock   "PSO"
  SrcPort   1
  Points   [30, 0; 0, -20]
  DstBlock   "Mux"
  DstPort   3
}
Line {
  SrcBlock   "Mux"
  SrcPort   1
  DstBlock   "ISL"
  DstPort   1
}
Line {
  SrcBlock   "Math\nFunction"
  SrcPort   1
  DstBlock   "v"
  DstPort   1
}
Line {
  SrcBlock   "Saturation"
```

```
  SrcPort   1
  Points   [5, 0]
  Branch {
    DstBlock     "Math\nFunction"
    DstPort     1
  }
  Branch {
    Points     [0, 70]
    DstBlock     "after sat"
    DstPort     1
  }
}
Line {
  SrcBlock   "ISL"
  SrcPort   1
  Points   [40, 0]
  Branch {
    DstBlock     "Saturation"
    DstPort     1
  }
  Branch {
    DstBlock     "before sat"
    DstPort     1
  }
}
      }
    }
    Block {
      BlockType       Sin
      Name       "Input"
      Position       [245, 130, 275, 160]
      SineType       "Time based"
      Amplitude       ".00000001"
      Bias       "-.00000001"
      SampleTime       "0"
    }
    Block {
      BlockType       Constant
      Name       "Particle\nLocation"
      Position       [245, 200, 275, 230]
      Value       "particle"
    }
```

```
    Block {
      BlockType       SubSystem
      Name       "Plant"
      Ports       [1, 2]
      Position       [435, 194, 535, 236]
      TreatAsAtomicUnit       off
      System {
Name "Plant"
Location [2, 74, 1022, 700]
Open off
ModelBrowserVisibility off
ModelBrowserWidth 200
ScreenColor "white"
PaperOrientation "landscape"
PaperPositionMode "auto"
PaperType "usletter"
PaperUnits "inches"
ZoomFactor "100"
Block {
  BlockType   Inport
  Name   "v"
  Position   [255, 255, 285, 265]
  Orientation   "up"
}
Block {
  BlockType   Fcn
  Name   "Fcn"
  Position   [160, 200, 220, 230]
  Orientation   "left"
  NamePlacement   "alternate"
  Expr   "(c1*u[1])+(c2*u[2])+((c3*u[3]^2)/((h+u[1])^"
"2))"
}
Block {
  BlockType   Mux
  Name   "Mux"
  Ports   [2, 1]
  Position   [305, 126, 310, 164]
  ShowName   off
  Inputs   "2"
  DisplayOption   "bar"
}
```

```
Block {
  BlockType   Mux
  Name   "Mux1"
  Ports   [3, 1]
  Position   [250, 194, 255, 236]
  Orientation   "left"
  NamePlacement   "alternate"
  ShowName   off
  Inputs   "3"
  DisplayOption   "bar"
}
Block {
  BlockType   Integrator
  Name   "z1_dot          z1"
  Ports   [1, 1]
  Position   [245, 65, 275, 95]
}
Block {
  BlockType   Integrator
  Name   "z2_dot      z2"
  Ports   [1, 1]
  Position   [160, 64, 190, 96]
}
Block {
  BlockType   Outport
  Name   "out"
  Position   [365, 73, 395, 87]
}
Block {
  BlockType   Outport
  Name   "x_bar"
  Position   [365, 138, 395, 152]
  Port   "2"
}
Line {
  SrcBlock   "z2_dot      z2"
  SrcPort   1
  Points   [20, 0]
  Branch {
    DstBlock     "z1_dot          z1"
    DstPort     1
  }
```

```
  Branch {
    Points     [0, 75; 70, 0]
    Branch {
      DstBlock       "Mux"
      DstPort       2
    }
    Branch {
      Points       [0, 60]
      DstBlock       "Mux1"
      DstPort       2
    }
  }
}
Line {
  SrcBlock   "Mux"
  SrcPort   1
  DstBlock   "x_bar"
  DstPort   1
}
Line {
  SrcBlock   "Fcn"
  SrcPort   1
  Points   [-20, 0; 0, -135]
  DstBlock   "z2_dot      z2"
  DstPort   1
}
Line {
  SrcBlock   "z1_dot          z1"
  SrcPort   1
  Points   [15, 0]
  Branch {
    DstBlock     "out"
    DstPort     1
  }
  Branch {
    Points     [-5, 0; 0, 55]
    Branch {
      DstBlock       "Mux"
      DstPort       1
    }
    Branch {
      Points       [0, 65]
```

```
      DstBlock       "Mux1"
      DstPort       1
    }
  }
}
Line {
  SrcBlock   "Mux1"
  SrcPort   1
  DstBlock   "Fcn"
  DstPort   1
}
Line {
  SrcBlock   "v"
  SrcPort   1
  DstBlock   "Mux1"
  DstPort   3
}
Annotation {
  Position   [324, 121]
}
      }
    }
    Block {
      BlockType       TransferFcn
      Name       "Reference Model"
      Position       [365, 126, 470, 164]
    }
    Block {
      BlockType       Sum
      Name       "Sum"
      Ports       [2, 1]
      Position       [595, 135, 615, 155]
      ShowName       off
      IconShape       "round"
      Inputs       "|+-"
      InputSameDT       off
      OutDataTypeMode       "Inherit via internal rule"
    }
    Block {
      BlockType       ToWorkspace
      Name       "error out"
      Position       [645, 170, 705, 200]
```

```
      VariableName       "er"
      MaxDataPoints       "inf"
      SampleTime       "-1"
      SaveFormat       "Array"
    }
    Block {
      BlockType       ToWorkspace
      Name       "plant out"
      Position       [645, 230, 705, 260]
      VariableName       "plant"
      MaxDataPoints       "inf"
      SampleTime       "-1"
      SaveFormat       "Array"
    }
    Line {
      SrcBlock       "Reference Model"
      SrcPort       1
      Points       [75, 0]
      Branch {
DstBlock "Sum"
DstPort 1
      }
      Branch {
Points [0, -20]
DstBlock " "
DstPort 1
      }
    }
    Line {
      SrcBlock       "Plant"
      SrcPort       1
      Points       [65, 0]
      Branch {
Points [0, 40]
DstBlock "plant out"
DstPort 1
      }
      Branch {
DstBlock "Sum"
DstPort 2
      }
    }
```

```
    Line {
      SrcBlock       "Sum"
      SrcPort        1
      Points        [0, 40]
      DstBlock       "error out"
      DstPort        1
    }
    Line {
      SrcBlock       "Plant"
      SrcPort        2
      Points        [10, 0; 0, 30; -255, 0]
      DstBlock       "Control"
      DstPort        3
    }
    Line {
      SrcBlock       "Particle\nLocation"
      SrcPort        1
      DstBlock       "Control"
      DstPort        2
    }
    Line {
      SrcBlock       "Control"
      SrcPort        1
      DstBlock       "Plant"
      DstPort        1
    }
    Line {
      SrcBlock       "Input"
      SrcPort        1
      Points        [0, 0; 5, 0]
      Branch {
Points [0, 55]
DstBlock "Control"
DstPort 1
      }
      Branch {
DstBlock "Reference Model"
DstPort 1
      }
    }
    Annotation {
      Name       "model out"
```

```
      Position       [671, 152]
    }
  }
}
```

A.7 Standard Deviation Function

psosd.m was used to calculate the standard deviation of the swarm for each iteration.

```
function out=psosd(in)
dims=size(in);
%out=((1/(dims(1)))
_*(sum(sum((in-repmat((sum(in')/dims(1))',1,dims(2))).^2))))^.5;
cog=sum(in')/dims(2);
dist=(sum((in-repmat(cog',1,dims(2))).^2)).^.5;
out=((1/dims(2))*sum(dist.^2)).^.5;
```