

Particle Swarm-Assisted State Feedback Control: From Pole Selection to State Estimation

Jiuguang Wang, Benjamin T. Brackett and Ronald G. Harley

Abstract—This paper presents automatic tools to optimize pole locations in state feedback control to satisfy performance specifications such as rise time, settling time, overshoot, and steady state error. Through the use of Particle Swarm Optimization (PSO), the advantages of such time-domain based controllers over frequency-domain controllers such as the Proportional-Integral-Derivative (PID) control is demonstrated, particularly for Multi-Input Multi-Output (MIMO) systems with a large number of states. A proof of concept problem involving the stabilization of a helicopter under the Rationalized Helicopter Model is presented and the results of the PSO auto-tuning, including the simultaneous design of a state observer and feedback controller, are supported by illustrative computer simulations.

Index Terms— Particle Swarm Optimization, Linear Systems, Pole Placement, State Feedback, State Estimation

I. INTRODUCTION

THE design of feedback controllers to satisfy performance specifications such as rise time, settling time, overshoot, and steady state error is well understood in the domain of classical control theory, utilizing frequency-domain approaches such as root locus, Bode, and Nyquist plots in conjunction with simple controllers such as the Proportional-Integral-Derivative (PID) control [1]. For Multi-Input Multi-Output (MIMO) systems, it is easier to work in the time-domain through the so-called state space representation since that with p inputs and q outputs, $q \times p$ transfer functions are needed to encode the equivalent information. State feedback control [2], also known as pole placement, involves the relocation of system poles to the left half s -plane in achieving stability in the closed-loop, with only the assumption that the system is controllable or observable. While the pole placement procedure is more straightforward than PID parameter tuning, it is unclear how the locations of closed-loop poles relate to the performance of the controller, especially in the transient state.

Without sophisticated counterparts to the frequency-domain based tuning methods, pole placement in the time-domain mainly consists of trial and error where a control engineer attempts several iterations of different pole locations until a desirable response is obtained. Traditional optimization techniques are unavailable as the optimization

objective can only be written as a combination of transient performance parameters (constants) where gradients cannot be computed. While we can cast this as a search problem, such an iterative search is highly dimensional and is only feasible for lower order problems. In a n -state linear system, for example, an optimization routine would have to search in a $2n$ -dimensional hyperspace, accounting for the real and imaginary parts of each pole; moreover, the design of an observer effectively doubles the search space. Most traditional optimization routines cannot handle such a high dimensional task.

In the existing literature, several previous works have considered using evolutionary algorithms for control design since these methods do not require explicit gradient information for optimization. For an overview of evolutionary algorithm in control engineering, see [3]. In particular, pole placement in [4] and [5] was formulated as a multi-objective optimization problem and solved with genetic algorithms (GAs). Our work is more comparable with [6] and [7], where GAs were used for robust pole placement and in nonlinear MIMO systems. More recently, Particle Swarm Optimization (PSO) was used to tune PI and PID controllers [8][9] and these works have shown that PSO is fast and reliable tool for control optimization, outperforming other evolutionary algorithms such as GAs.

The main contribution of this paper is the use of PSO for optimizing pole locations in satisfying transient and steady state performance specifications, which extends the previous work on PSO-tuned PID controllers to the time-domain. The PSO algorithm, developed by James Kennedy and Russell C. Eberhart in 1995 [10][11], is inspired by the paradigm of social interaction and searches the solution hyperspace by a population of moving particles. Compared to other computational intelligence techniques such as neural networks and genetic algorithms, PSO performs extremely well in high dimensional optimization problems, and we show that it is effective in finding optimal pole locations in state feedback control.

The rest of this paper is organized as follows. Section II gives a brief overview of the time-domain control design using state feedback control, state estimation, and the hand-tuning of pole locations for transient and steady state performance. Section III discusses the Particle Swarm Optimization algorithm and formulates the problem of pole placement using PSO, including details on design issues such as the construction of the fitness function and system simulation. Section IV demonstrates the PSO optimized state feedback control for the case of a helicopter stabilization under the

Jiuguang Wang and Ronald G. Harley are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia, 30332, USA. Emails: j.w@gatech.edu, ronald.harley@ece.gatech.edu

Benjamin T. Brackett is with the Georgia Tech Research Institute, Atlanta, Georgia, 30332, USA. Email: ben.brackett@gttri.gatech.edu

Color versions of one or more of the figures in this paper are available online at <http://ieeexplore.ieee.org>

Rationalized Helicopter Model (RHM). We conclude with Section V and discuss future research directions.

II. STATE FEEDBACK CONTROL

Consider the linear time-invariant (LTI) system

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \quad (1)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \quad (2)$$

where $\mathbf{x}(t)$ is the state vector, $\mathbf{u}(t)$ is the input vector, and $\mathbf{y}(t)$ is the output vector. If the system is controllable, i.e., if it is possible to find some input function $\mathbf{u}(t)$ that will transfer the initial state $\mathbf{x}(t_0)$ to the origin at some finite time, then the full state feedback law

$$\mathbf{u} = -\mathbf{K}\mathbf{x}(t) \quad (3)$$

gives a stable closed-loop system. The controllability of a linear system can be easily verified by constructing the controllability matrix

$$\mathbf{R} = [\mathbf{B} \quad \mathbf{A}\mathbf{B} \quad \dots \quad \mathbf{A}^{n-1}\mathbf{B}] \quad (4)$$

such that if \mathbf{R} has a full rank, then the system is controllable.

If the internal state of the system is not directly accessible, the separation principle can be used to design a state observer such that $\mathbf{x}(t)$ can be indirectly obtained from the output $\mathbf{y}(t)$. The output feedback method with a Luenberger observer has the dynamics

$$\dot{\hat{\mathbf{x}}}(t) = \mathbf{A}\hat{\mathbf{x}}(t) + \mathbf{B}\mathbf{u}(t) - \mathbf{L}(\hat{\mathbf{y}}(t) - \mathbf{y}(t)) \quad (5)$$

$$\hat{\mathbf{y}}(t) = \mathbf{C}\hat{\mathbf{x}}(t) \quad (6)$$

which can be used if the system is observable, i.e., if the observability matrix

$$\mathbf{Q} = [\mathbf{C} \quad \mathbf{C}\mathbf{A} \quad \dots \quad \mathbf{C}\mathbf{A}^{n-1}]^T \quad (7)$$

has a full rank. Both the full and output feedback methods are well-established in linear systems theory and detailed derivation and designs can be found in [2]. The state feedback approach described here is also fully valid for the regulation problem, where we seek to find a set of controls $\mathbf{u}(t)$ to track some reference output $\mathbf{y}_{ref}(t)$.

The key design issue in state feedback control is the selection of the feedback gains \mathbf{K} and \mathbf{L} , which are directly computed from the closed loop poles. One attractive feature of the state feedback method is that as long as the poles are placed in the left side of the s -plane, the closed-loop response will yield a zero steady state error. Compared to other frequency-domain controllers design, the pole placement procedure is more straightforward in this respect. The lack of graphical tools such as root locus and Bode plots, however, means that it is significantly more difficult to tune the state feedback controller for transient state performance, such as rise time, settling time, and overshoot. Control engineers are often forced to rely on intuition for the tuning of these parameters. There are two general heuristics on the pole locations:

- 1) As poles are placed further to the left side of the s -plane, i.e., approaching $-\infty$, the transient performance

improves significantly, while poles near the origin lead to a slower transient response.

- 2) The penalty of placing poles further to the left side is two fold: it leads to overwhelmingly high magnitude in the control input $\mathbf{u}(t)$ and the resulting system is drastically more sensitive to external disturbances.

In making these tradeoffs, a conventional approach is to rely on optimal control, particularly the linear-quadratic regulator (LQR) [12], in which a cost functional of the form

$$J = \int_0^\infty (\mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \mathbf{R} \mathbf{u}) dt \quad (8)$$

can be used to express the accuracy vs. control effort tradeoff. The LQR feedback law

$$\mathbf{u}(t) = -(\mathbf{R}^{-1} \mathbf{B}^T \mathbf{P}(t)) \mathbf{x}(t) \quad (9)$$

gives the optimal control with respect to the cost functional, where $\mathbf{P}(t)$ is the solution to the algebraic Riccati equation. While it is equivalent to the state feedback law in (3), the fundamental problem of using optimal control to design for transient performance is that the available cost heuristics are in terms of constant parameters, e.g., settling time and overshoot, which do not give the gradients information necessary to derive the feedback law. Without knowing how to express these costs in terms of the \mathbf{Q} and \mathbf{R} weight matrices, the design for transient and steady-state performance still requires manual tuning, a time-consuming process often taking a few iterations before a desirable response can be obtained.

III. PARTICLE SWARM OPTIMIZATION

A. Overview

Particle Swarm Optimization, developed by James Kennedy and Russell C. Eberhart in 1995, is a stochastic, population-based evolutionary computing technique inspired by the paradigm of social interaction. Instead of the gradient information utilized by most other optimization algorithms, PSO relies on the trajectories of a group of potential solutions called "particles", which traverse the solution hyperspace in \mathbb{R}^n . At each time instance, the quality of the solutions obtained by each particle is assessed by a fitness function and each individual fitness value is shared across neighboring particles. Individually, the particles record their own best fitness value and the associated solution, as well as the group's best fitness and solution. Over time, the group as a whole is drawn stochastically towards the global optimum.

The flow of the PSO algorithm can be described by Algorithm 1, where

- m is the number of particles
- x is the position of the particles, i.e., the potential solutions
- v is the velocity of the particles
- f_{lbest} and x_{lbest} is the best fitness value and position for individual particles
- f_{gbest} and x_{gbest} is the best fitness value and position for the group

Algorithm 1 *PSO – SEARCH*

```
1: Initialize  $m$  PSO particles
2: while iteration  $k <$  maximum iterations count do
3:   for each particle  $j \in [1 \dots m]$  do
4:      $f \leftarrow$  PSO-FITNESS( $x[j]$ )
5:     if  $f < f_{lbest}[j]$  then
6:        $f_{lbest}[j] \leftarrow f$ 
7:        $x_{lbest}[j] \leftarrow x[j]$ 
8:     end if
9:     if  $f < f_{gbest}$  then
10:       $f_{gbest} \leftarrow f$ 
11:       $x_{gbest} \leftarrow x[j]$ 
12:    end if
13:     $v[j, k + 1] = wv[j, k] + c_1\phi_1(x_{lbest}[j, k] - x[j, k]) +$   

      $c_2\phi_2(x_{gbest}[j, k] - x[j, k])$ 
14:     $x[j, k + 1] = x[j, k] + v[j, k + 1]$ 
15:  end for
16: end while
```

PSO-FITNESS is the fitness function used to assess the quality of the solutions obtained and is constructed according to the optimization objective. Since the goal of this paper is the application of PSO to the pole placement problem, we refrain from lengthy discussions on intricate details of the algorithm itself and instead refer the readers to [13].

A critical component of the PSO algorithm is the position and velocity update equations in lines 13-14. There are three design parameters:

- 1) ϕ_1 and ϕ_2 are uniformly distributed random variables bounded in $[0, 1]$, used to generate the trajectories.
- 2) c_1 and c_2 are the acceleration constants which guide each particle toward the best solution found by itself as well as the group.
- 3) w is the inertia constant, used to provide balance over exploration and exploitation.

Typically, the particle velocities are also bounded within a range $[-v_{max}, v_{max}]$ and when the velocity update equation returns a result violating this bound, the appropriate upper and lower limits are used as the velocity instead.

The power of PSO lies in its simplicity. Unlike optimal control and other optimization algorithms, PSO does not assume an advanced mathematical background and its construction involving the two update equations and some housekeeping operations is simple and straightforward. By understanding that the first term in the velocity update equation is used for global exploration and the last two terms are used for local search, the PSO convergence response can be easily tuned to yield better performance. For parameter selection and variations on the PSO algorithm, see [13].

B. Poles and Particles

In formulating the problem of optimal pole placement with PSO, an important design issue is how the pole locations should be represented as particle positions. While in theory, the closed-loop poles can be arbitrarily placed anywhere in the left side of the complex plane, from an optimization point

of view, the dimensions of the search problem can be reduced by restricting the poles to be complex conjugates, i.e., of the form

$$p_i = a \pm bj \quad (10)$$

With this restriction, the solution hyperspace is reduced by half, since only one set of values for a and b need to be determined for each pair of poles. This implies that for a state space system of n states, the hyperspace is contained within \mathbb{R}^n . In the case of output feedback, where the poles for the Luenberger observer are designed simultaneously, the observer poles double the size of the hyperspace to \mathbb{R}^{2n} .

In our approach, we encode each particle to include the set of closed-loop poles for the system, i.e., a set of a and b values from (10). As the PSO algorithm progresses and the particles traverse the search space, the poles associated with each particle is used to simulate the control response.

C. Fitness Function

The fitness function in PSO plays an important role in that it directly influences the trajectories of the particles. In the pole placement problem, the fitness function evaluates the pole locations, i.e., the particle's position and returns a fitness value with consideration of the user-defined performance specifications. The fitness function in PSO is equivalent to the objective function in traditional optimization algorithms, but similar to other evolutionary algorithms, the fitness function is optimized without explicitly computing any gradients.

The fitness function used in PSO-SEARCH utilizes five individual fitness measures of the transient and steady state responses:

- 1) Steady state error: the difference between the system output and the reference in steady state
- 2) Settling time: the time required to settle to within 5% of the steady state value
- 3) Overshoot: the amount exceeding the steady state value on the signal's initial rise
- 4) Rise time: the time required for travel from the 10% to the 90% level
- 5) Maximum input limit: absolute value of the the maximum amplitude of the input signal

There are a variety of other metrics to evaluate the quality of the system output, but the five measures above are the most general and frequently used. The user is asked to enter five target values for the five parameters and at each fitness evaluation, the actual performance of the system is compared to the target specifications. If the actual system performance is within the specified limit, then a fitness value of zero is assigned to the parameter; otherwise, the percent error of actual vs. target specs is calculated.

The final fitness value returned by the fitness function is the sum of all individual fitness values, assuming that the weighting factors for the parameters are equal (penalized equally). For MIMO systems, the multiple input and output signals are evaluated separately and the fitnesses are summed together to produce the final value. The importance of a particular signal or performance parameter can be expressed

by using a larger weighting factor. While PSO allows multi-objective optimization where there are multiple (potentially competing) objectives, we found that single-objective PSO is sufficient to solve the pole placement problem.

D. System Simulation

An important observation in the construction of the fitness function is the evaluation of the actual system performance. For both the transient and steady state parameters, responses can only be obtained by simulating the system, which is much more computationally intensive than the PSO algorithm itself. The Runge-Kutta method [14] is typically used for control systems simulations, but in our approach, we chose the Euler method [15], a basic method of numerical integration for ordinary differential equations, to obtain the control response, where

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h(\mathbf{A}\mathbf{x}_n + \mathbf{B}\mathbf{u}) \quad (11)$$

for $\mathbf{x}(t_0) = \mathbf{x}_0$. This is a typical simplification, since for a linear time-invariant state space system, the Euler method produces accurate results yet remains fast and simple to implement.

Also, a number of numerical algorithms exist for pole placement, with the Kautsky algorithm [16] being the most widely used. In the MATLAB environment, the Kautsky algorithm was implemented as the *place()* function in the Control Systems Toolbox, taking in the system matrices and a list of desired pole locations as inputs and outputting a gain matrix for use in the control law. For the full state and the output feedback cases, $K = \text{place}(\mathbf{A}, \mathbf{B}, P)$ and $L = \text{place}(\mathbf{A}', \mathbf{C}', P)'$ gives the plant and observer feedback gains, respectively.

IV. EXAMPLE - HELICOPTER STABILIZATION

A helicopter stabilization design was chosen as a proof of concept problem in demonstrating the PSO-based state feedback design. The Rationalized Helicopter Model (RHM) [17] is a well-studied nonlinear dynamical model of single-rotor helicopters. Modeled after the Westland Lynx helicopters, the RHM accounts for a four-blade semi-rigid main rotor and rigid body. The equations governing the helicopter motions are complex and the open loop dynamics are unstable throughout the flight envelope, exhibiting highly cross-coupled and nonlinear response.

A. Dynamics

In this design, the dynamical model is taken from [18], where a linear system

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \quad (12)$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) \quad (13)$$

approximates the small-perturbation rigid motion of the aircraft about hover. The state vector $\mathbf{x}(t)$ contains eight states:

- θ - pitch attitude
- ϕ - roll attitude
- p - roll rate (body-axis)

- q - pitch rate (body-axis)
- ξ - yaw rate
- v_x - forward velocity
- v_y - lateral velocity
- v_z - vertical velocity

The output $\mathbf{y}(t)$ contains six variables made up of four controlled outputs

- H_t - heave velocity
- θ - pitch attitude
- ϕ - roll attitude
- ψ_t - heading rate

and two additional body-axis measurements

- p - roll rate
- q - pitch rate

The four blade angles serve as the inputs to the helicopter

- u_1 - main rotor collective
- u_2 - longitudinal cyclic
- u_3 - lateral cyclic
- u_4 - tail rotor collective

The $\mathbf{A}, \mathbf{B}, \mathbf{C}$ matrices for the RHM model can be obtained from [19] as a MATLAB script.

Roughly, the main rotor collective input controls the lift by rotating the rotor blades. The longitudinal and lateral cyclic inputs control the longitudinal and lateral motions by varying the blade angles. The tail rotor balances the torque generated by the main rotor, preventing the aircraft from spinning and also gives the desired lateral motion. This model assumes that the dynamics are decoupled when in reality, it is highly coupled, which leads to non-minimum phase characteristics at certain operating points. For more discussions on the dynamics, see [18].

B. Control design

Previous studies on the RHM model consisted of robust control designs and disturbance rejections to reduce the effects of atmospheric turbulence. While PSO can be used for this purpose, we instead focus on the simpler task of optimizing for transient state performance, highlighting the high dimensional search capabilities of the algorithm - for the eight-state RHM model, the PSO algorithm searches for the optimal output feedback controller design in \mathbb{R}^{16} .

Consider the scenario of a helicopter motion in some non-zero initial state, and the control objective is to stabilize the system to the origin, i.e., to reach a state of equilibrium where the vector sum of all forces and all moments is equal to zero. The performance targets are defined as follows:

- Steady State Error: 0.1
- Settling Time: 2s
- Maximum input: 100

Since the control objective is not to track a step reference, other transient performance metrics such as overshoot and rise time are not included.

After the construction of the controllability and observability matrices, it is verified that the plant is both controllable and observable. A series of hand tuned simulations are first conducted using a duration of 5 seconds and a time step of

0.01. Iteratively, pole locations are first identified to satisfy the steady state error requirement, then settling time, and finally maximum input. After five iterations in approximately three minutes, the response in Fig. 1 and 2 are obtained, where only the maximum input still does not fall under the desired level. Further simulations either do not reduce $\mathbf{u}(t)$ or violate the first two constraints. For comparison, this reasonable tuning effort produces a response with

- Steady State Error: 0.0285
- Settling Time: 1.9933s
- Maximum input: 176.5133

Next, the identical system is given to a PSO-based auto-tuner with a set of standard parameters:

- Number of Particles (m): 10
- Max number of Iterations (k): 100
- Inertia Constant (w): 0.7
- Acceleration Constants (c_1 and c_2): 1
- Maximum Particle Velocity (v_{max}): 2

The particle positions are contained within a $2 \times 6 \times 10$ matrix, accounting for the two axes (real and imaginary), six pairs of complex-conjugate poles (three for the plant and three for the observer) and the ten particles. All parameters in the multiple inputs and outputs are penalized equally in the fitness calculations. The plots corresponding to the PSO-tuned simulations are shown in Fig. 3 and Fig. 4.

The PSO-tuned design is simulated using a Intel Core 2 Duo 2.0 GHz computer with 2 GB of memory. Finishing in approximately 15 seconds (or comparably, less time than a single hand-tuned iteration), the search returned a response with

- Steady State Error: 0.00031038
- Settling Time: 2.7790s
- Maximum input: 70.8414

In comparison with the hand-tuned results, the greater (but still acceptable) settling time reduces the maximum input by 150%.

Various other combinations of PSO parameters are also experimented with in attempts to further improve the system response. Extending the number of particles, maximum number of iterations, as well as various constants yield slight variations in the output, most of which show trade-offs between the maximum input and settling time. Overall, the original set of parameters used appears to give the best performance with the shortest computation time. It is noted that the exact procedure in tuning the PSO parameters is still unknown and there is vast literature on the topic with numerous solutions on how variations to the update equations and parameters can lead to faster convergence.

While we did not compare the performance of our PSO-based control design to the alternative methods discussed in the literature review, we can offer some simple analysis on the expected performance difference based on studies on the PSO algorithm itself. As stated previously, most optimization routines used for control design are evolutionary algorithms, which do not require the explicit computation of gradients. In the existing literature, PSO has been extensively

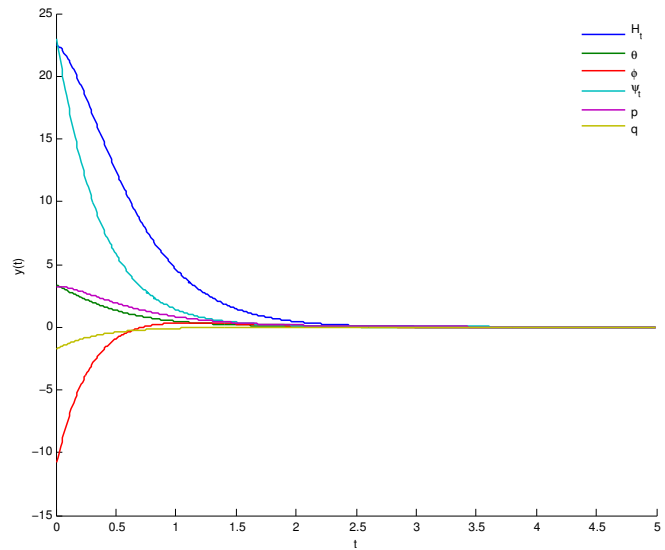


Fig. 1. Hand-tuned pole placement simulation showing the six outputs stabilizing to zero in 1.9933 seconds.

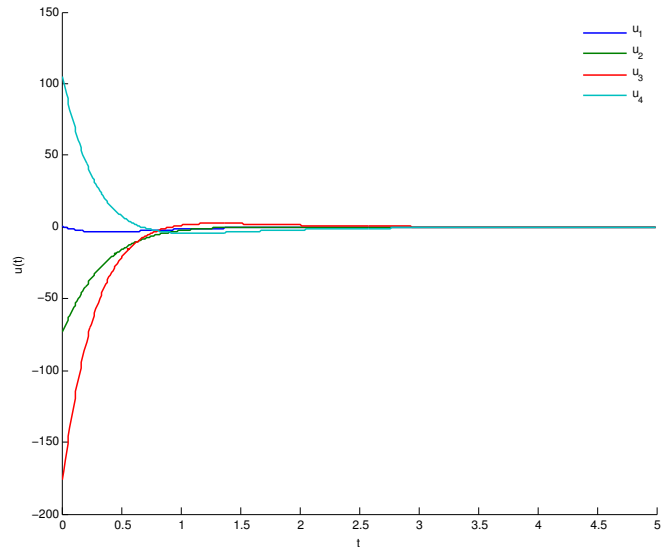


Fig. 2. Hand-tuned control inputs corresponding to Fig. 1 with a maximum value of 176.5133 (u_3), exceeding the max input constraint by 43%.

studied against other evolutionary algorithms, and the results show that it consistently outperforms genetic algorithms and similar routines. We refer the readers to [20] and [21] for discussions on the philosophical and performance comparisons.

V. CONCLUSIONS

In this paper, Particle Swarm Optimization (PSO) was implemented to search for the optimal pole locations in state feedback control to satisfy transient and steady state performance specifications such as rise time, settling time, overshoot, and steady state error. A proof of concept problem involving the stabilization of a helicopter was presented and the results of the PSO auto-tuning were supported by

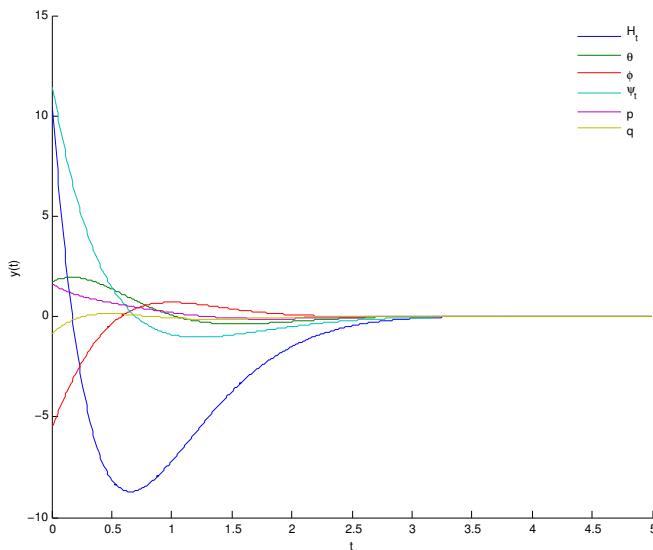


Fig. 3. PSO-assisted pole placement simulation showing that the six outputs stabilize to zero in 2.7790 seconds.

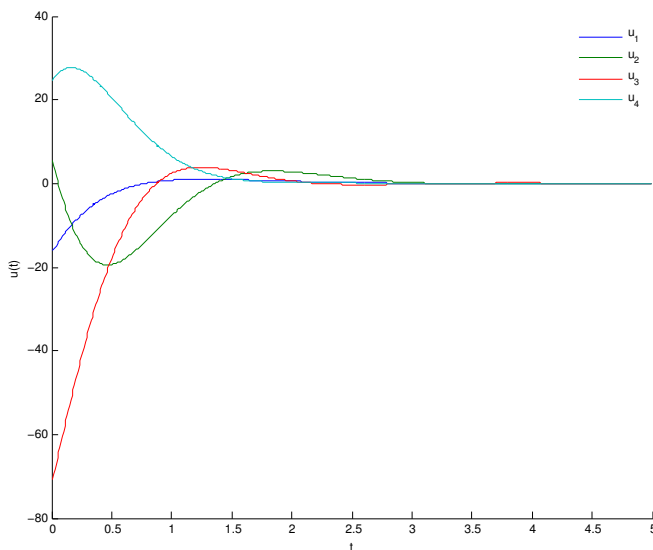


Fig. 4. PSO-tuned control inputs corresponding to Fig. 3 with a maximum value of 70.8414 (u_3), under the given max input constraint.

illustrative computer simulations.

PSO is a powerful optimization tool in the realm of control theory, particularly for the time-domain state space systems, where performance tuning has not been well-studied in the past. Possible future work include extensions to nonlinear and adaptive control theory, where the techniques such as feedback linearization and gain scheduling can be used in conjunction with PSO. In addition, the variety of improved PSO algorithms that have surfaced in the past decade could potentially be used to produce better results, and it would be interesting to see how the PSO parameter tuning can be simplified with these new algorithms.

ACKNOWLEDGMENTS

This work was done as a part of the Summer 2008 course *Computational Intelligence for Electric Power* at Georgia Tech. The authors are grateful to Dr. Magnus Egerstedt, Brian Smith, Michael Laughter, and Douglas Brooks for reviewing the manuscript and for their helpful comments. We also thank the two anonymous reviewers for their comments and suggestions on an earlier version of this manuscript.

REFERENCES

- [1] P. Cominos and N. Munro, "PID controllers: recent tuning methods and design to specification," *IEE Proceedings on Control Theory and Applications*, vol. 149, no. 1, pp. 46–53, Jan 2002.
- [2] W. Brogan, *Modern Control Theory*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 1991.
- [3] P. Fleming and R. Purshouse, "Evolutionary algorithms in control systems engineering: a survey," *Control Engineering Practice*, vol. 10, no. 11, pp. 1223–1241, 2002.
- [4] C. Fonseca and P. Fleming, "Multiobjective optimal controller design with genetic algorithms," *International Conference on Control*, vol. 1, pp. 745–749, March 1994.
- [5] G. Sanchez, M. Villasana, and M. Strefezza, "Multi-objective pole placement with evolutionary algorithms," *Lecture Notes in Computer Science*, vol. 4403, p. 417, 2007.
- [6] J. Ouyang and W. Qu, "Robust pole placement using genetic algorithms," in *International Conference on Machine Learning and Cybernetics*, vol. 2, 2002.
- [7] R. Dimeo and K. Lee, "Boiler-turbine control system design using a genetic algorithm," *IEEE Transaction on Energy Conversion*, vol. 10, no. 4, pp. 752–759, 1995.
- [8] W. Qiao, G. Venayagamoorthy, and R. Harley, "Design of optimal PI controllers for doubly fed induction generators driven by wind turbines using particle swarm optimization," *2006 International Joint Conference on Neural Networks*, pp. 1982–1987, 0-0 2006.
- [9] Z.-L. Gaing, "A particle swarm optimization approach for optimum design of PID controller in AVR system," *IEEE Transaction on Energy Conversion*, vol. 19, no. 2, pp. 384–391, June 2004.
- [10] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," *1995 IEEE International Conference on Neural Networks*, vol. 4, pp. 1942–1948, Nov/Dec 1995.
- [11] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, Feb 2002.
- [12] A. E. Bryson and Y.-C. Ho, *Applied Optimal Control: Optimization, Estimation, and Control*. Washington, DC: Hemisphere Publishing Corporation, 1975.
- [13] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.
- [14] J. Butcher, *Numerical methods for ordinary differential equations*. Wiley, 2003.
- [15] U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM: Society for Industrial and Applied Mathematics, 1998.
- [16] J. Kautsky, "Robust pole assignment in linear state feedback," *International Journal of Control*, vol. 41, no. 5, pp. 1129–1155, May 1986.
- [17] G. Padfield, "A theoretical model of helicopter flight mechanics for application to piloted simulation," *RAE Technical Report*, vol. 81048, 1981.
- [18] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*. Wiley, 2005.
- [19] S. Skogestad. (2008, September) Rationalized helicopter model. [Online]. Available: <http://www.nt.ntnu.no/users/skoge/book/2nd.edition/matlab.html>
- [20] R. Eberhart and Y. Shi, "Comparison between genetic algorithms and particle swarm optimization," *Lecture Notes in Computer Science*, pp. 611–618, 1998.
- [21] P. Angeline, "Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences," *Lecture Notes in Computer Science*, pp. 601–610, 1998.