Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

Particle Swarm Optimization for PID Tuning of a BLDC Motor

Alberto A. Portillo
UTSA Department of Electrical Engineering
University of Texas Health Science Center
PM&R
San Antonio, TX

Michael Frye, Ph. D.
Department of Engineering
University of the Incarnate Word
San Antonio, TX

Chunjiang Qian, Ph. D.
UTSA Department of Electrical Engineering
University of Texas at San Antonio
San Antonio, TX

Abstract— **This paper applies a particle swarm optimizer to tune the PID gains for a brushless DC motor. This motor is used in a milling machine that moves in coordination with three additional motors to carve out transtibial and transfemoral prosthetic sockets. Three motors move in the X, Y, and Z coordinate axes, while one motor rotates in a clockwise/counterclockwise rotation. The BLDC motor of interested is responsible for controlling the movement along the X axis. Performance improvements using the PSO PID gains are analyzed in the motor controlling the X axis by comparing the performance of the motor obtained by an optimizer program provided by the manufacturer of the milling machine and the brushless DC motor model developed in this paper.**

***Keywords*—Particle Swarm Optimization, PID tuning, biomedical application**

I. INTRODUCTION

The digital motion controller used in the milling of transtibial and transfemoral prosthetic sockets is a Galil's DMC-1842 which controls four Parker OEM675 motor amplifiers. The four motors being used are Parker SM series brushless DC motor (BLDC). The milling machine that uses each of these elements carves out the shape of a patient's lower extremity amputation into a foam block. From this foam carve, a prosthetist melts a sheet of plastic and vacuum fits the plastic over the foam from which a prosthetic socket is made from various materials. The demands of the foam carve require the plastic mold to be as accurate as possible so that an air tight seal is created when the amputee is fitted into the socket. Also, the length of time that is spent carving out the socket needs to be as fast as possible with out compromising the quality of the carve in order to give the prosthetist optimal amount of time to create a socket, fit the patient, and correct for errors in a time efficient manor to give the best quality care the patient expects from the clinician. This paper develops a motor model by way of a transfer function and modeled in Matlab in order to test and implement motor improvements for time and accuracy of the carve. The Windows Servo Design Kit (WSDK) is a specific software package developed by Galil to run the DMC-1842 motion controller. The WSDK has an Auto Crossover Frequency (ACF) automated tuning feature that allows the user to automatically find optimized PID gains. This tuning method was used to find the PID gains and then implemented in the transfer function to be modeled in Matlab. To compare the performance characteristics of the ACF tuning method provided by the WSDK Galil software, particle swarm optimization (PSO) is investigated to find specific parameters and tested to find optimized PID gains for the motor controller. PSO is a heuristic algorithm first introduced in 1995 as an alternative method to find optimized proportional-integral-derivative values by James Kennedy and Russell Eberhart [1]. The PSO was used to find optimized PID gains and the values were implemented in the transfer function to compare with the open loop modeling and root-locus modeling of the optimized PID gains that were obtained using the WSDK ACF tuning function. The results show a smaller overshoot, a faster settling time, and improved stability as shown in the root-locus analysis.

II. MOTOR MODEL

The transfer function of the motors movement needed to be found in order to perform a comparison analysis of the WSDK ACF PID values versus the PSO PID optimized values. By finding the individual transfer functions and gain values from each of the individual components used to drive the motor, the transfer function was found and modeled in Matlab. A step impulse was used to drive the transfer function so that the percent overshoot, rise time, and settling time could be analyzed. The root-locus was used to further determine the stability of the transfer function.

A. Calculation the PID Values

The WSDK's ACF tuning function was used to find the optimal PID values for several important reasons. (1) It is a general tuning tool to find optimal PID values for general motor movement. (2) It is easy to use and autonomous once the parameters are established. (3) It provides reasonably

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

consistent PID values over several implementations. The parameters of the WSDK were set to move the motor 50 encoder counts/radian (cnts/rad). From this movement, the PID values were generated and recorded for use in the motion model. The PID values generated from the WSDK ACF were (40.50/7.52/381) respectively for P, I, and D.

B. Elements of the Motor Model

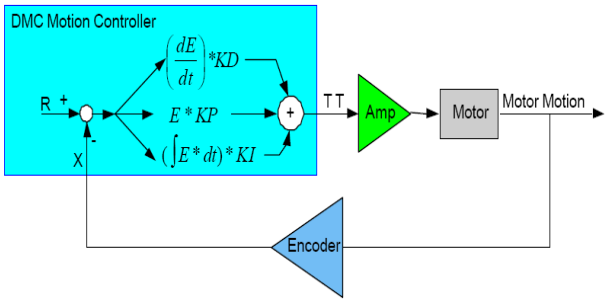

Figure 1: Components of the Motor Model [5]

Figure 1 shows the elements for which the transfer function of the motor model was built for the DMC motion controller, the amplifier, the motor, and the encoder. The DMC motion controller is further separated into three parts: the PID digital filter, the zero order hold (ZOH), and the digital to analog converter (DAC).

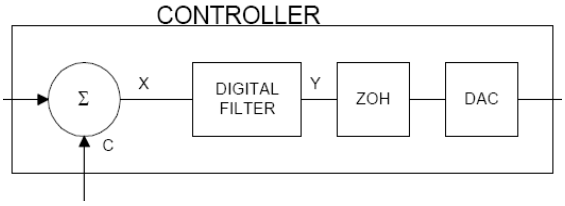

Figure 2: Elements of the DMC Motor Controller [6]

The PID digital filters transfer function is realized in Equation 1. The values for P, I, and D are obtained from the WSDK ACF tuning function,

$$PID = \frac{Ds^2 + Ps + I}{s}. \quad (1)$$

The ZOHs transfer function in the controller is designed to control the sampling time at which the motor command is updated inside the controller,

$$H(s) = \frac{1}{(1+\frac{sT}{2})} = \frac{1}{(1+\frac{s(0.001)}{2})} = \frac{2000}{s+2000}. \quad (2)$$

The gain of the DAC component is found to be:

$$K_d = \frac{20V}{2^{16}cnt} = 0.003 \text{ v/cnt}. \quad (3)$$

Equation 4 shows the combined transfer functions and gains from Equations 1 – 3 to create the transfer function for the DMC motion controller,

$$\frac{0.6Ds^2 + 0.6Ps + 0.6I}{s^2 + 2000s}. \quad (4)$$

The OEM675 amplifier was selected to driver the SM233BE series parker BLDC servo motor. The selection for this amplifier was based on the manufacturers' recommendation of the appropriate amplifier to drive the specific motor [4]. The output gain of the amplifier is 1.2 amp/volt. The BLDC motor is able to provide accuracy at high speed with the embedded encoders and hall effect sensors keeping track of the motors internal position. The BLDC motor is also able to provide a large amount of toque upwards of 11.3 lb.-in. continuous torque. This large torque provides sufficient amount of force to drive loaded objects and provide the accuracy and speeds needed to move multiple objects.

The efficient integration of the motor and the encoders located on the shaft of the motor required the individual modeling of the motors movement and the encoder. The gain of the encoder is found to be 636, based on Equation 5. The variable *N* is defined by the product specifications to be 1000 pulses per revolution (ppr),

$$K_f = \frac{4N}{2\pi} = \frac{4(1000)}{2\pi} = 636. \quad (5)$$

The position of the motor is calculated using the values shown in Table 1 and implemented in Equation 6 shown below,

Table 1: SM233BE System Parameters

Parameters	Symbol	Units	SM233BE
Resistance	Ra	Ohms	2.58
Rotor Inertia	J	kg-m^2	1.30E-04
Torque Constant	Kt	Nm/Amp peak	0.208
Viscous Damping	B	Nm/Krpm	3.78E-03

$$\frac{\theta(s)}{V(s)} = \frac{\frac{K_t}{R_a * j}}{s\left[s + \frac{1}{j}\left(B + \frac{K_t^2}{R}\right)\right]}. \quad (6)$$

Combing Equations 4 – 6 yields Equation 7, the open loop characteristic transfer function of the motion model. The PID

values from the WSDK ACF tuning function provided the values of (40.50/7.52/381) respectively for P, I, and D,

$$\frac{4.154\text{E}^5 s^2 + 4.524\text{E}^7 s + 1.0769\text{E}^8}{s^3 + 2127 s^2 + 2.537\text{E}^5 s}. \quad (7)$$

C. Motor Model Results

The closed loop step response of equation with PID values of (40.50/7.53/381) is shown in Figure 1. From the graph, several important calculations are found that are used to compare to the PSO PID values in the same closed loop transfer function shown in Equation 7. These calculations are peak time, settling time, and percent overshoot are shown in Table 2.

Table 2: WSDK ACF Performance Response

Performance Characteristics	WSDK ACF
Peak Time	0.0375 sec.
Settling Time	0.0923 sec.
Percent Overshoot	8.80%

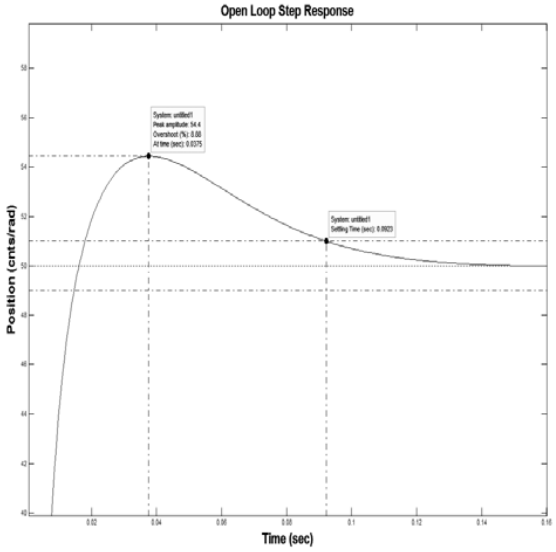

Figure 1. Open Loop Step Response.

The root-locus method was also used to determine the rate of stability of the open-loop response of Equation 7. Figure 2 shows the root-locus response. The WSDK ACF PID values achieved stability by root-locus examination at the complex poles of -36.36±24i where the gain (K) is equal to $4.154\text{x}10^5$.

Table 3: WSDK ACK Root Locus Performance

Performance Characteristics	PSO PID
Gain (K)	$2.977\text{x}10^5$
Complex Poles (S)	-6.67±6.53i

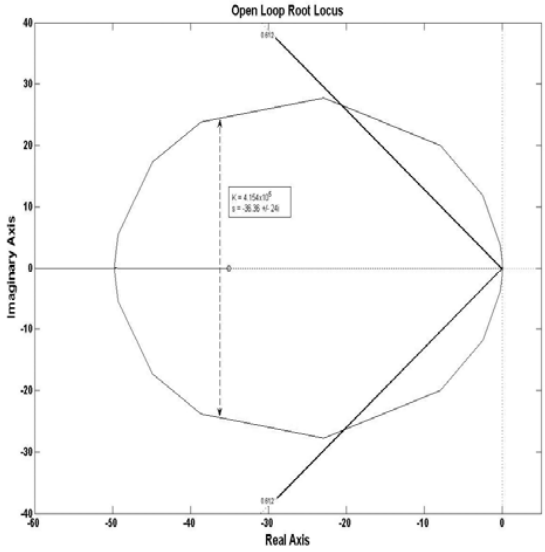

Figure 2. Open-Loop Root Locus.

III. Particle Swarm Optimization

PSO is a stochastic swarm optimization technique as described by Kennedy and Eberhart [1]. Kennedy and Eberhart discuss that a swarm of bees will work together to find an area with the most food. Each individual bee in the swarm searches a random area and finds the area for the most abundant food. The individual bee will then relay this location to the entire swarm and compare this individual location of food to the other locations reported by the other bees. Based upon the input, the bee will either keep the location because the other bees have not returned a location with a greater food density, or will gravitate towards the location with the higher density in comparison to its own location. The individual bee does a random search, this time with the location of the highest density in mind and tries to find a location of higher food density. If the bee finds a higher density while doing the random search, the bee relays this new information to the rest of the swarm. Each bee has a new location for the highest food density and begins to gravitate to the new location. Eventually, the entire swarm will collectively gather at the location with the highest density of food.

Kennedy and Eberhart [1] bridges the biological concept of the swarm behavior to an engineering focus by using the idea that in nature, a swarm will find an optimal solution. In engineering, the swarm concept could be adapted as yet another tool to find optimized values in a multivariable environment, such as finding optimized PID values. Robinson and Rahmat-Samii [2] further expand on the PSO discussion by bringing attention to several key elements that make the PSO an effective engineering optimizing tool. The *particles* in

the swarm are the individual elements in the swarm responsible for moving to there personal best values (*pbest*) and the swarms best values (*gbest*) all the while continually searching there current *position* to monitor for better values than what the individual has. The individuals' *position* is the location given a specific boundary for which to search in. Evaluation of the position is performed through a *fitness* function that returns the optimal solution.

I) Number of *particles*
The number of *particles* was assigned to 100 with the intent that this would allow for a large number of individual elements to better explore and converge on the optimal PID gains. Research by Robinson [2] and Carliel [3] suggest that the higher number of *particles* gives an improved exploration for optimal values when compared to the number of *particles* at 20 or 30. The compromise between the higher number is that the higher values takes longer to compute than at 20 or 30 particles.

II) PID search space
In order to find the optimal values for the proportional, integral, and derivate elements, three ranges were established based upon the output from the WSDK ACF tuning. The PID values resulting from auto tuning resulted in PID values of 40.5, 7.52, and 381 respectively. The search space for which the particles worked in were set to 50, 10, and 450 in order to provide an effective window to perform a search that would be similar to the PID values found using the WSDK ACF tuning method. The search space also establishes the boundaries to which the fitness function is to be evaluated.

III) Fitness function
The *pbest* and *gbest* are initial assigned random values in the search space. For each iteration, The *pbest* and *gbest* values are compared to the current location. If the current location has better optimized values than the current *pbest*/*gbest* values, the fitness function returns a numerical value that is used to evaluate a new velocity and the new values replace the old *pbest/gbest* values.

IV) Number of trials
The number of trials was set to 50 iterations for each *particle*. This number was established to give each *particle* an opportunity to successfully find optimal PID values over multiple trials.

V) Inertial weight (0.9 – 0.01)
The internal weight, ω, as suggested by Robinson [2] and Carliel [3] was initial set to 0.9 to provide a global search which is less influenced by *pbest* and *gbest*. As the number of iterations increased toward 50, the internal weight reduced towards 0.01 resulting in a condensed area that put more emphasis on the activities of *pbest* and *gbest*.

VI) C1 & C2 values
The social factors of C1 and C2 determine the amount of emphases the *particles* velocity is effected by *pbest/gbest*. C1 is set to 1 and C2 is set to 2 thus putting more emphases on *gbest*.

VII) Velocity
In order for the velocity (Vn) to be calculated, the fitness functions of *pbest* and *gbest* needed to be evaluated. The rand(•) element in the equation provides the function a sense of natural behavior found in nature [2].

$$Vn_{new} = w * Vn_{old} + C1 * rand()(pbest - xn) + C2 * rand()(gbest - xn)$$

VIII) Movement
The movement of the *particle* is accomplished by adding the new velocity to the current location.

$$Xn = Xn + t * Vn_{new}$$

IV. PSO PID RESULTS

The PSO PID function produced gain results of (47.50/2.31/273.08). The new PID values were implemented in Equation 4 and a new Equation 7 was calculated and simulated through Matlab. The new PID results are shown in Table 4.

Table 4: PSO PID Performance Results

Performance Characteristics	PSO PID
Peak Time	0.0225 sec.
Settling Time	0.0823 sec.
Percent Overshoot	5.64%

The results for the root-locus analysis are shown in Figure 3 and Table 5.

Table 5: PSO PID Root Locus Analysis

Performance Characteristics	Auto WSDK
Gain (K)	4.154×10^5
Complex Poles (S)	-6.67±6.53i

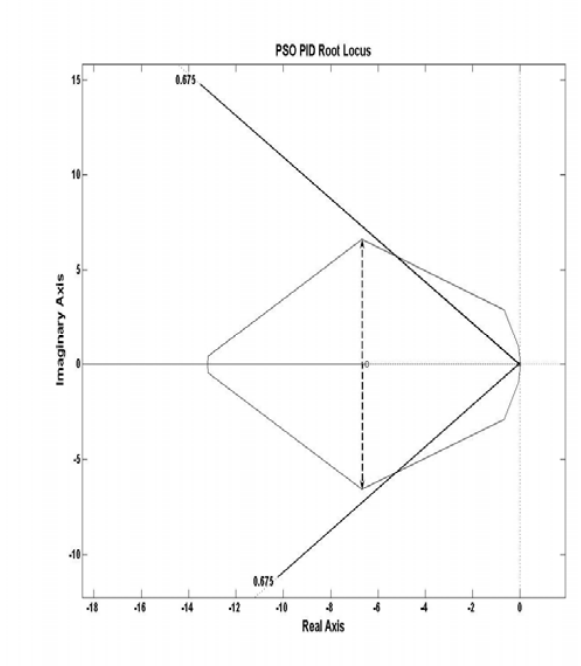

Figure 3. PSO PID Root Locus.

V. RESULTS

The resulting closed-loop step response parameters using the PID values derived from the PSO algorithm were found to be an improvement from the PID values found using the WSDK ACF. Comparing the values of Table 2 and Table 4 show the improvements in peak time, settling time, and percent overshoot. The relative percent difference equation is used to determine a statistical improvement in all three performance characteristics for the closed loop step response. These performance improvement values are shown in Table 6.

Table 6: Step Response Performance Improvements

Performance Characteristics	% Improvement
Peak Time	50
Settling Time	11.45
Percent Overshoot	43.76

The root-locus analysis also showed a faster rate for stability of the model and a lower gain (K). Using the PID values obtained with the PSO, the new K was 2.977×10^5 resulting in complex poles at -6.67±6.53i. By comparing Table 3 and Table 5, the performance improvements can be seen that the PSO PID converges towards stability much faster than the WSDK ACF PID values.

Upon successful results with the PSO PID over the WSDK ACF PID values, the two PID values were implemented in the actual system to determine if the modeled performance actually had an improvement in the real system. Figure 4 shows the two PID values implemented in the physical system. When comparing the output graphs of the two optimizing functions, the implemented PSO PID values show a faster settling time with a faster rise time, little to no overshoot and a fast settling time. The ACF WSDK PID values result in a much longer rise time, a large overshoot, and a substantial settling time when compared to the PSO PID. It is shown that the response in the physical system is consistent with the improved calculations in the closed loop step response and the root locus analysis between the two PID optimizing functions.

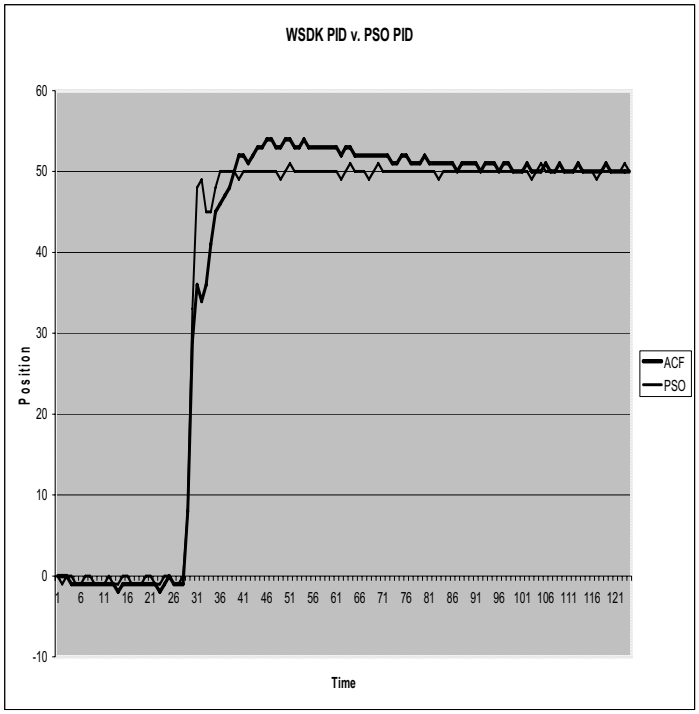

Figure 4: PSO v. WSDK comparison

ACKNOWLEDGMENT

We would like to thank the support and facilities of the Andrew Gitter GAIT Laboratory at the University of Texas Health Science Center.

REFERENCES

[1] James Kennedy, R. E. (1995). Particle Swarm Optimization. IEEE International Conference on Neural Networks, Perth, Australia, IEEE Service Center, Piscataway, NJ.

[2] Jacob Robinson, Y. R.-S. (2004). "Particle Swarm Optimization in Electromagnetics." IEEE Transactions on Antennas and Propagation **52**(2): 397-407.

[3] Anthony Carlisle, G. D. (2001). AN Off-The-Shelf PSO. Proceedings of the 2001 Workshop on Particle Swarm Optimization, Indianapolis, In.

[4] Compumotor (1998). OEM670X/675X, Parker Hannifin Corporation.

[5] Galil Motion Control, Inc.. Application Note #3413: Manual Tuning Methods. Rocklin, CA

[6] Tal, J. (1994). STEP-BY-STEP DESIGN OF MOTION CONTROL SYSTEMS. Mountain View, CA, Galil Motion Control, Inc.

[7] Portillo, A. (2008). Using the Particle Swarm Optimizer to Solve for PID values for a BLDC Motor. Electrical Engineering. San Antonio, University of Texas at San Antonio: 60.