

**A COMPARISON OF IDENTIFICATION
TECHNIQUES FOR ROBOT CALIBRATION**

By

ROBERT JOHN HORNING

Submitted in partial fulfillment of the requirements
for the degree of Master of Science

Project Advisor: Dr. Wyatt Newman

Department of Electrical Engineering

CASE WESTERN RESERVE UNIVERSITY

August, 1998

A COMPARISON OF IDENTIFICATION TECHNIQUES FOR ROBOT CALIBRATION

Abstract

By

ROBERT JOHN HORNING

This work presents a comparison of four different methodologies for identifying the calibration parameters of a Motoman P-8 robot located at Sandia National Laboratories. This robot is a 6-dof, all revolute robot with one of the most popular configurations. The robot was modeled using a modified Denavit and Hartenberg technique that utilized Hayati's notation to represent transformations between nearly parallel joint axes. The four identification techniques considered were the Steepest Gradient method, the Monte Carlo method, the Guided Evolutionary Simulated Annealing method, and the Circle-Point method. These four techniques were evaluated against four different data sets. The first data set consisted of a very friendly test function used to verify that the algorithms were working properly. The second data set consisted of simulated, ideal data that had no unmodeled effects. The third data set was also simulated, but had random noise added to the three Cartesian coordinates of each data point. The last data set consisted of calibration data that was generated from the Motoman P-8 robot. This evaluation shows that of the four techniques, the Circle-Point method is the fastest and most consistently accurate method of identification. Thus, it is a highly attractive technique whenever the data acquisition process satisfies the requirements.

DEDICATION

I dedicate this work to the memory of my father, who, along with my mother, made everything possible.

ACKNOWLEDGEMENTS

There are many people, without whom, this work would not have been possible. First and foremost, I would like to thank my adviser, Dr. Wyatt Newman, for providing me with direction and advice, along with his ongoing support. I want to thank Sandia National Laboratories for providing the inspiration for this work, along with the real world robotic hardware and calibration data that this work is based on. In addition, I would like to thank the Center for Automation and Intelligent Systems Research (CAISR) for providing the facilities and environment I used during the course of this work.

I also want to recognize and say thanks to all the other Mechatronics students who have helped me along the way. A special thanks goes out to Mark Doring for his assistance with the robot and computer hardware in the CAISR lab, and Craig Birkhimer, whose efforts have been directly incorporated into this work. I would like to thank coworkers Brian Barnt and Greg Conley, Brian for his programming assistance, and Greg for letting me use his compiler.

On a personal note, I want to thank my dear friend Judy for encouraging, supporting, and motivating me to complete this work. I also want to thank my family for their continued support. Lastly, I want to especially thank my parents, to whom this work is dedicated, for their never-ending love, support and encouragement. Without them, I wouldn't be where I am today.

Cleveland, Ohio

ROBERT HORNING

August, 1998

TABLE OF CONTENTS

DEDICATION.....	iii
ACKNOWLEDGEMENTS	iv
LIST OF FIGURES AND TABLES	vii
LIST OF SYMBOLS.....	viii
1.0 INTRODUCTION.....	1
2.0 KINEMATIC MODEL	5
3.0 IDENTIFICATION TECHNIQUES	8
3.1 “Goodness” measurement	8
3.2 Steepest Gradient search algorithm	9
3.3 Monte Carlo search algorithm	10
3.4 Guided Evolutionary Simulated Annealing algorithm	11
3.5 Circle-Point algorithm.....	14
4.0 EXPERIMENTAL RESULTS	16
4.1 Test function.....	16
4.1.1 Corrupted model for testing	17
4.1.2 Steepest Gradient results on test function	17
4.1.3 Monte Carlo results on test function	18
4.1.4 GESA results on test function	19
4.2 Simulated noise-free data set.....	20
4.2.1 Steepest Gradient results on simulated noise-free data	20
4.2.2 Monte Carlo results on simulated noise-free data	21
4.2.3 GESA Results on simulated noise-free data.....	21
4.2.4 Circle-Point Results on simulated noise-free data	22

4.3	Simulated data with noise.....	23
4.3.1	Steepest Gradient results on simulated data with noise	23
4.3.2	Monte Carlo results on simulated data with noise	24
4.3.3	GESA results on simulated data with noise	24
4.3.4	Circle-point results on simulated data with noise	25
4.4	Calibration data set.....	26
4.4.1	Steepest Gradient results on calibration data	27
4.4.2	Monte Carlo results on calibration data	28
4.4.3	GESA results on calibration data	29
4.4.4	Circle-point results on calibration data	30
5.0	CONCLUSIONS	32
	REFERENCES	36

LIST OF FIGURES AND TABLES

Figure 1: Schematic of the Motoman P-8 robot	1
Table 1 : Nominal kinematic model	7
Table 2 : Nominal (factory) kinematic model	17
Table 3: Corrupted kinematic model for tests on simulated data	17
Table 4: Steepest Gradient results on test function	18
Table 5: Monte Carlo results on test function	19
Table 6: GESA results on test function	19
Table 7: Steepest Gradient results on simulated noise-free data	21
Table 8: Monte Carlo results on simulated noise-free data	21
Table 9: GESA results on simulated noise-free data	22
Table 10: Circle-Point results on simulated noise-free data	22
Table 11: Steepest Gradient results on noisy simulated data.....	23
Table 12: Monte Carlo results on noisy simulated data	24
Table 13: GESA results on simulated noisy data	25
Table 14: Circle-Point results on noisy simulated data	26
Table 15: Steepest Gradient results on calibration data.....	27
Table 16: Nominal model subtracted from Steepest Gradient results on calibration data.....	28
Table 17: Monte Carlo results on calibration data.....	28
Table 18: Nominal model subtracted from Monte Carlo results on calibration data.....	29
Table 19: GESA results on calibration data	29
Table 20: Nominal model subtracted from GESA results on calibration data	30
Table 21: Circle-Point results on calibration data	30
Table 22: Nominal model subtracted from Circle Point results on calibration data.....	31

LIST OF SYMBOLS

a_i	Length of link i (D-H notation)
d_i	Offset length of link i (D-H notation)
α_i	Twist angle of link i (D-H notation)
$\theta_{\text{home},i}$	Home angle offset for joint i (D-H notation)
β_i	Additional rotation parameter for nearly parallel axes (Hayati notation)
$Error^2$	Squared euclidean distance error
$X_{\text{model}}, Y_{\text{model}}, Z_{\text{model}}$	Model predicted Cartesian coordinates of end effector
$X_{\text{actual}}, Y_{\text{actual}}, Z_{\text{actual}}$	Measured Cartesian coordinates of end effector
rms_err	Average root mean squared error of model over entire data set
n	Number of data points in data set
Par	One of the 27 parameters in a kinematic model
$offset$	Small parameter offset value used in calculating the gradient
Par_{child}	One of the 27 parameters of a child
Par_{parent}	Corresponding parameter of a child's parent
p	Random number in the range of 0 to 1
$weight$	Weight value used to control the size of the search
ΔPar	Delta parameter value used to calculate the gradient
Δrms_error	Delta rms_error value used to calculate the gradient
$rms_error_{(Par+offset)}$	Rms error as calculated with parameter Par increased by $offset$
$rms_error_{(Par-offset)}$	Rms error as calculated with parameter Par decreased by $offset$
ΔPar_max	Maximum random offset that can be added to a parameter

<i>const</i>	Constant used for manually adjusting the reference magnitude of <i>weight</i>
<i>kick_mult</i>	Multiplier used to “kick” the search when it becomes stuck
<i>weight_mult</i>	Multiplier used to reduce the search area as the algorithm gets closer to the solution
<i>rms_error_{lowest}</i>	Current best rms error found
<i>y_{new}</i>	Rms error of the best child
<i>y_{best}</i>	Rms error of the best parent found thus far
<i>t₁</i>	Temperature value used to control the rate of convergence
<i>y_{child}</i>	Rms error of the child
<i>t₂</i>	Temperature value used to control the rate of acceptance
<i>M</i>	Average number of children in each family
<i>N</i>	Total number of families
<i>acc_i</i>	Number of accepted children in family <i>i</i>
<i>sum_acc</i>	Total number of accepted children in the population

1.0 INTRODUCTION

The problem of robot calibration has been the study of much research over the years. In this work, a comparison of four different calibration techniques is presented. This work focused on the calibration of a Motoman P-8 robot located at Sandia National Laboratories. This robot, shown schematically in Figure 1, is a 6-dof, all revolute robot with one of the most popular configurations. The “waist” axis controls a base rotation, the “shoulder” axis is orthogonal to the waist axis, the “elbow” axis is parallel to the shoulder axis, and rotations of the forearm, wrist, and toolflange, respectively, comprise a (nominally) spherical wrist. It is the manufacturer’s intent that the shoulder axis be identically perpendicular to the waist axis (albeit, non-intersecting), that the elbow axis be identically parallel to the shoulder axis, that the forearm axis be perpendicular to the elbow axis (though non-intersecting), and that the last 3 joint axes intersect at a common point.

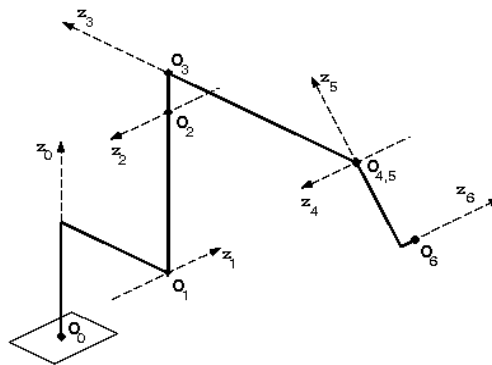


Figure 1: Schematic of the Motoman P-8 robot

This particular robot was retrofitted with a Cimetrix, Inc. PC-based controller[10]. The Cimetrix controller incorporates a DSP-based motion-control card that achieved static

positioning accuracy with respect to joint-angle commands within several encoder counts. The user interface allowed ease of programming data-acquisition applications in “C”[1].

In addition, an SMX¹ Tracker 4000 coordinate measurement machine (CMM) was used to accurately measure the position of the robot’s end effector in the laboratory reference frame. The SMX Tracker 4000 is also a PC-controlled system. It consists of a laser source and laser interferometer (for measuring range) within a pan/tilt head (for measuring azimuth and elevation). The laser source retro-reflects off of a corner-cube mirror mounted at the robot’s tool frame. As the robot moves, the SMX system continuously tracks the moving mirror target, and in this manner, the interferometer is able to keep track of absolute distance. The precision of the interferometer was 0.16 microns. Azimuth and elevation measurements are limited by the precision of the pan/tilt head angular sensors, which is constant with respect to spherical coordinates at 0.25 arc-seconds. This precision corresponds to a variable precision with respect to Cartesian coordinates (with maximum precision obtainable for large azimuth and elevation changes). Roughly, for the range of motions considered in this work, and relative robot and sensor locations, a Cartesian precision from the SMX system within 25 microns at a distance of 5 meters can be expected[1].

The four calibration techniques considered in this work consisted of a gradient based iterative technique[3], two evolutionary techniques, and a non-iterative technique. The two evolutionary methods were a Monte Carlo search algorithm[5][**Error! Reference source not found.**], and the Guided Evolutionary Simulated Annealing (GESA) algorithm presented by Nyberg and Pao[2]. The non-iterative technique was the Circle-Point method presented by

¹ SMX Incorporated, 222 Gale Lane, Kennet Square, PA 19328, www.smxcorp.com

Stone[4]. All of these algorithms were programmed in “C”, and are described in detail in section 3.0.

These four techniques were evaluated under four different scenarios. First, to verify that the algorithms were functioning properly, they were run on a very friendly test function that had only one minimum, which was located at the correct solution, and had smooth sides. This function is described in Section 4.1. The Circle Point algorithm was not evaluated against this function because its structure is not applicable.

Once the algorithms were determined to be functioning properly, they were evaluated against a simulated, ideal data set. This data set was generated using a simulated robot defined by the manufacturers stated robot parameters. The data set was created by swinging each joint of the simulated robot through an arc while holding all the other joints constant. Since this data set was generated mathematically, it had no unmodeled effects, thus making it a perfect data set. This data set was also used to verify the correctness of the Circle Point algorithm.

Next, the four algorithms were evaluated against a simulated data set similar to the one described above, but had random noise added to its Cartesian coordinates. Lastly, the algorithms were run on a calibration data set. This data set consisted of actual measurements recorded from the real robot. The calibration data was also collected by rotating one joint of the robot at a time, while all the other joints were held constant.

Through these evaluations, it will be shown that the Circle-Point algorithm is the superior identification technique when the data acquisition process satisfies the algorithm's requirements.

2.0 KINEMATIC MODEL

The kinematic representation used in this work as the basis for calibration is the widely used representation that is defined by Denavit and Hartenberg [11], in which transformations between successive link coordinate frames are expressed in terms of 3 fixed parameters and one joint variable. For all-revolute robots, such as the P-8, the fixed parameters are a_i , d_i and α_i for link i . The Denavit-Hartenberg (D-H) notation defines a reference zero for the joint variable, but since this reference zero typically does not conform to the physical sensor zero, one must define an additional parameter, $\theta_{\text{home},i}$, for each joint to describe the home-angle offsets.

One failing of the D-H representation is that the parameters are discontinuous with respect to perturbations of nearly parallel joint axes. This discontinuity problem is addressed by Hayati[12] with the substitution of an additional rotation parameter (β_i) for nearly parallel axes transformations and suppression of the parameter d_i to zero.

Newman, *et al.*[1] propose one further refinement of the representation applicable to sequential, nearly intersecting joint axes. According to D-H notation, each link coordinate frame is defined in terms of z axes collinear with joint axes, and x axes defined collinear with the common normal between successive z axes. (Unfortunately, details on how to construct and label link axes vary inconsistently among different authors; see, e.g., [11], [13], [14], and [15]). If sequential joint axes do not intersect, the x axis defined along the common normal is directed from the proximal z axis towards the distal z axis. If the z axes intersect, the

direction of the common normal has a +/- ambiguity. In parameter identification, one cannot assume that joints intended to intersect will actually intersect, and one must identify the best-fit a_i parameter (distance between the axes). Since a_i can vary +/- about the nominal zero value, the corresponding D-H model can experience reversals of the corresponding defined x-axis direction.

Newman, *et al.* propose to define the direction of axis \mathbf{x}_i unambiguously as the cross product of the direction vector of axis \mathbf{z}_{i-1} into axis \mathbf{z}_i . This definition is consistent with the D-H definition but resolves the ambiguity for intersecting axes. The ambiguity under this definition re-appears for parallel axes, but this case is better handled by Hayati notation. In addition to resolving the ambiguity (for non-parallel axes), this definition eliminates the parametric discontinuity problem for nearly intersecting axes. Consequently, negative a_i parameters can result (which is never the case with D-H notation).

The resulting mixed D-H and Hayati notation fully represents the (ideal) kinematics of the robot in terms of successive transformations. Sequential frame assignments based on pairs of z axes describe all but the first and last transforms: from the CMM reference frame to the base frame of the robot, and from the z axis of the tool-flange rotation to the corner-cube measurement target. These two transformations are dependent on how the CMM, the robot and the mirror target are mounted. Thus, they are extrinsic parameters, not part of the intrinsic robot model. In [1], they defined the base-frame parameters d_1 and $\theta_{\text{home},1}$ with respect to the CMM reference frame, though these parameters will differ for different mounting instances of the same robot. Since their reference frame z axis was nearly parallel to the robot's waist axis, they described the transformation from the sensor frame to the base

frame in terms of Hayati parameters. For the final (tool) frame, they define the origin with respect to the corner-cube target (which is deliberately offset from the joint-6 axis) and orientation such that α_6 is identically 90 deg. The manufacturer's data gives a nominal model, summarized in Table 1. The identification problem consists of finding a set of 27 parameters (all those except marked by "x" or "#" in Table 1) that best fit the measured data.

Frame	a_i mm	d_i mm	α_i rad	β_i rad	$\theta_{\text{home},i}$ rad
0	*	x	*	*	*
1	-350	*	$\pi/2$	x	*
2	1100	x	π	0	$\pi/2$
3	-300	0	$\pi/2$	x	$\pi/2$
4	0	-1200	$\pi/2$	x	π
5	0	0	$\pi/2$	x	π
6	*	*	$\pi/2^\#$	x	*

*→extrinsic parameter—application specific

x→parameter not used for this frame

#→parameter defined at specified value

Table 1 : Nominal kinematic model

3.0 IDENTIFICATION TECHNIQUES

3.1 “Goodness” measurement

For all of the search algorithms presented below, some measure of relative “goodness” was needed to evaluate how accurate a particular solution was. The value used for this purpose was the total root mean squared error. This rms error was calculated in the following way for any given data set. Each data point consisted of a set of six joint angles, and a corresponding set of three Cartesian coordinates. The Cartesian coordinates represent the actual location of the end effector when the robot was positioned at the corresponding six joint angles.

The rms error was calculated by going through all of the data points one at a time. For each data point, the forward kinematics of the D-H parameter model being evaluated was calculated using the six joint angles of the data point. The Cartesian coordinates of the predicted end effector location were then taken from the forward kinematics model. A squared Euclidean distance error was next calculated for this data point as shown in equation 3-1.

$$Error^2 = (x_{model} - x_{actual})^2 + (y_{model} - y_{actual})^2 + (z_{model} - z_{actual})^2 \quad (3-1)$$

These squared errors were then added together to create a total squared error value for the entire data set. Afterwards, equation 3-2 was used to calculate the rms error for the model when evaluated against the entire data set.

$$rms_err = \sqrt{\frac{\sum Error^2}{n}} \quad (3-2)$$

This rms error value was the measure of goodness used to compare the different D-H models.

3.2 Steepest Gradient search algorithm

The first identification technique considered was the Steepest Gradient search technique[3]. The specific algorithm used here was programmed in the following manner. The algorithm started its search with a D-H parameter model specified by the user. The rms error of this initial parameter set was calculated. Next, a gradient vector was calculated for the parameter set. This was done by calculating the ratio in equation 3-3 for each of the 27 parameters. This ratio was calculated by first adding a small offset to one parameter at a time, and recalculating the rms error of the new model. Then the same offset value was subtracted from the original parameter value, and the rms error again recalculated. These values were then subtracted respectively as shown in equation 3-3 to generate the gradient value for this parameter. After all the parameters had been evaluated, the resulting gradient vector was normalized.

$$\frac{\Delta rms_error}{\Delta Par} = \frac{rms_error_{(Par+offset)} - rms_error_{(Par-offset)}}{(Par + offset) - (Par - offset)} \quad (3-3)$$

Once the gradient vector had been calculated, the algorithm started to move in the direction of the gradient vector. After taking a step in the gradient direction, a new rms error was calculated. This value was then compared to the previous best rms error. If it was smaller, the algorithm took another step in the same direction. This process was repeated either until the rms error of the next step was larger than that of the last, or until 50 total steps had been taken.

After having moved as far as possible in the gradient direction, the step size was evaluated and adjusted. If the number of steps taken was less than two, the step size was cut in half for the next iteration. If the number of steps taken was larger than twenty, the step size was doubled. In this way, the algorithm could control how fast it moved in the gradient direction based on the results of the last iteration.

Subsequent to adjusting the step size, a new gradient vector was calculated and the process repeated. This process went on either until a minimum rms error value had been reached, or until the search was terminated manually.

3.3 Monte Carlo search algorithm

The Monte Carlo search used in this work was programmed in the following manner. The algorithm started its search with a D-H parameter model specified by the user. This model was used as the parent. To start, the rms error of this parent was calculated. Next, five children were created from this parent. Each child was created by adding random offsets to each of the parent's 27 D-H parameters. After the creation of each child, the child's rms error value was calculated. Next, the six family members were compared to determine which one had the lowest rms error. This family member then became the parent for the next generation and the process was repeated. This cycle repeated until either a minimum accepted rms error value was found, or the search was terminated manually.

One aspect of an evolutionary search such as this is that as the search progresses, getting closer and closer to a solution, the search area should get progressively smaller. If this is not done, the search may skip over the minimal solution. This was accomplished by using a

weight value to control the range of random offsets that were used to create the children in each generation. The formula shown in equation 3-4 was used to generate the random offsets for each parameter,

$$Par_{child} = Par_{parent} + p \times weight \times \Delta Par_max \quad (3-4)$$

where Par_{child} was one of the 27 parameters of the new child, Par_{parent} was the corresponding parameter of the parent, p was a random number in the range of -1 to $+1$, $weight$ was the weight value used to control the size of the search, and ΔPar_max was the maximum random offset that could be added. The weight value was recalculated each iteration using equation 3-5,

$$weight = const \times kick_mult \times weight_mult \quad (3-5)$$

where $const$ was a constant used for manually adjusting the reference magnitude of the weight, $kick_mult$ was a value used to “kick” the search when it was stuck, and $weight_mult$ was a multiplier used to reduce the size of the search as the algorithm got closer to the solution. The $kick_mult$ value was adjusted by keeping track of the number of iterations that have taken place without a better solution being found. When this number became greater than 20, the $kick_mult$ value was halved, thus having the size of the search area. Once a new solution was found, $kick_mult$ was set back to one. The $weight_mult$ value was adjusted by monitoring the searches progression. When the rms error value became an order of magnitude better, the $weight_mult$ multiplier was reduced by an order of magnitude.

3.4 Guided Evolutionary Simulated Annealing algorithm

The GESA algorithm used in this work was programmed as follows. Like the Steepest Gradient algorithm and the Monte Carlo algorithm, the GESA algorithm started its search from a D-H parameter model specified by the user. This search technique used three families

($N = 3$), with an average number of children per family of five ($M = 5$). To start off, the algorithm generated three parents. This was done by taking the initial D-H parameter set provided by the operator and adding or subtracting small random offsets to each of the D-H parameters. Once these parents had been generated, their relative “goodness” was calculated and saved. The parent that had the best rms error value was recorded as being the best solution thus far.

Once the algorithm was initialized, the main search loop began. First, five children were generated in each family. The rms error of each child was then calculated. These values were then compared to determine which child was the best. The best child of the family was then compared to the parent. If the child was better than the parent, the child was saved as the parent for the next generation. Even if the child was not better than the parent was, it might have been saved as the parent for the next generation if it was “pretty good”. “Pretty good” was defined by equation 3-6.

$$\text{Exp}\left[\frac{-(y_{new} - y_{best})}{t_1}\right] > p \quad (3-6)$$

where y_{new} was the rms error of the best child, y_{best} was the rms error of the best parent thus far, t_1 was a temperature value that was used to control the rate of convergence, and p was a random number between zero and one. When y_{new} was much larger than y_{best} , the exponent was very small, reducing the probability that it would be greater than p . When y_{new} was close to y_{best} , The exponent was close to one, creating a higher probability that it will be greater than p . In this way, the better a child was, the more likely it was to become the parent for the next generation, even if it wasn't as good as its parent.

Once the parents for the next generation had been determined, the next step was to allocate how many children each family would have for the next generation. This was accomplished by starting with the first family, and evaluating equation 3-7 for each child in the family.

$$\text{Exp}\left[\frac{-(y_{child} - y_{best})}{t_2}\right] > p \quad (3-7)$$

In equation 3-7, y_{child} was the rms error of the child, y_{best} was the rms error of the best parent found up to that point, t_2 was a temperature value that was used to control the rate of acceptance, and p was a random number between zero and one. If the exponential term was greater than p for a child, the number of children accepted in this family was incremented by one. This process was repeated for each family. After counting the number of accepted children in each family, these numbers were added together to find the total number of children accepted in the population. With this information, the number of children allocated to each family in the next generation was determined by equation 3-8.

$$M \times N \times \frac{acc_i}{sum_acc} \quad (3-8)$$

In equation 3-8, M was the average number of children in each family, N was the total number of families, acc_i was the number of accepted children in family i , and sum_acc was the total number of accepted children in the population. This had the effect of allocating all the available children each generation, but giving more to the families that were more successful in the previous generation, and less to the families that were less successful.

Once the allocation process was complete, the next iteration of the search was begun by randomly creating new children for each family in the manner described above. This process

was repeated until a solution with an rms error less than the minimum error tolerance was found, or until the search was manually terminated.

3.5 Circle-Point algorithm

The three algorithms presented thus far are general search techniques that can be applied to a variety of problems. The circle-point method takes a more specific approach to the problem of calibrating a revolute-joint robot[4]. This approach simplifies the data fit necessary to derive the robot parameters by deriving the parameters for only one joint at a time. The Circle-Point identification program used in this work was written by Newman, *et al.*[1].

To start, the data used for calibration needed to be collected in a particular way. This involved rotating one joint at a time through a number of angles, and measuring the end effector location at each pose. By rotating only one joint while all the other joints remain fixed, some geometric properties of the data are known. To start with, all of the points generated from rotating only one joint must lie in a plane. Thus, the analysis could be started by mathematically fitting a plane to the collected data. It is also known that the direction of the joint axis must be normal to this plane. Thus, once the plane that the data lies in had been determined, the direction of the joint axis was immediately known.

At this point, while the direction of the joint axis was known, its location was still undetermined. Something else, however, could be said about the data. Since the data was collected by rotating a single joint, all the points must lie on a circular arc. Thus, a circle could be mathematically fitted to the data points within the previously determined plane. Once the equation for the circle that contained the data points was known, the center of this

circle could be calculated. Since the joint axis must pass through the center of this circle, the origin of the joint axis was then known, along with its direction.

This process could then be repeated for each joint of the robot. By comparing the location and direction of each two adjacent joint axes, the kinematic transformation matrix from one joint axis to the next could be written.

4.0 EXPERIMENTAL RESULTS

4.1 Test function

Before the algorithms could be applied to the various data sets for a comparative analysis, they needed to be tested to verify that they were programmed correctly. One possible proof is based on the theory that if an algorithm is started sufficiently close to the correct solution (within the same valley) the search technique should be able to converge to the correct solution. Unfortunately, determining how close is “close enough” is not a trivial task. Because of this, another method was devised.

Instead of simply trying to start the algorithms “close enough” to the correct solution, a much friendlier function was used to test the algorithms. The function used is shown in equation 4-1,

$$error = (\|\mathbf{f} - \mathbf{f}_0\|)^2 \quad (4-1)$$

where \mathbf{f} was a vector containing all 27 of the kinematic model parameters, and \mathbf{f}_0 was a similar vector containing the nominal kinematic model shown in Table 2. This function is looks like a 27 dimensional “bowl” with no local minima. Thus, no matter where the search algorithm starts in function space, it should converge to the correct solution, since the function has no local minima.

Units: mm & radians

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0469	0.0008	-0.0133	4310.0793	0.0000
1	-0.0768	1.5708	0.0000	-350.0000	-684.7848
2	1.5708	3.1416	0.0000	1100.0000	0.0000
3	1.5708	1.5708	0.0000	-300.0000	0.0000
4	3.1416	1.5708	0.0000	0.0000	-1200.000
5	-3.1416	1.5708	0.0000	0.0000	0.0000
6	5.7633	1.5708	0.0000	50.8000	-184.1500

Table 2 : Nominal (factory) kinematic model

4.1.1 Corrupted model for testing

Because the iterative identification techniques required some initial starting point, a corrupted kinematic model was created based on the nominal factory model. The corrupted model was created by modifying only two of the 27 kinematic parameters. The Theta 2 home angle offset was modified by adding 0.25 degrees, and the Theta 3 home angle offset was modified by subtracting 0.25 degrees. This yielded the corrupted model shown in Table 3.

Units: mm & radians

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0469	0.0008	-0.0133	4310.0793	0.0000
1	-0.0768	1.5708	0.0000	-350.0000	-684.7848
2	1.5752	3.1416	0.0000	1100.0000	0.0000
3	1.5664	1.5708	0.0000	-300.0000	0.0000
4	3.1416	1.5708	0.0000	0.0000	-1200.000
5	-3.1416	1.5708	0.0000	0.0000	0.0000
6	5.7633	1.5708	0.000	50.8000	-184.1500

Table 3: Corrupted kinematic model for tests on simulated data

4.1.2 Steepest Gradient results on test function

The Steepest Gradient search algorithm was applied to the test function described above. It was started out with the corrupted model shown in Table 3. On the test function, this corrupted parameter set had an error value of $3.8e-05$ with respect to the test function. The search converged to the parameter set shown in Table 4, which has an error value of $9.37e-12$

with respect to the test function. This algorithm terminated on its own when it had converged within a minimum error value of $1.0e-11$. As expected, the algorithm converged very quickly when applied to the ideal test function, taking only 4 seconds. A comparison of the results in Table 4 with the known correct model from Table 2 shows that the algorithm has in fact converged to the known correct solution.

Units: mm & radians
 rms_error = $9.367999e-012$

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0469	0.0008	-0.0133	4310.0793	0.0000
1	-0.0768	1.5708	0.0000	-350.0000	-684.7848
2	1.5708	3.1416	0.0000	1100.0000	0.0000
3	1.5708	1.5708	0.0000	-300.0000	0.0000
4	3.1416	1.5708	0.0000	0.0000	-1200.000
5	-3.1416	1.5708	0.0000	0.0000	0.0000
6	5.7633	1.5708	0.0000	50.8000	-184.1500

Table 4: Steepest Gradient results on test function

4.1.3 Monte Carlo results on test function

The Monte Carlo search was next evaluated against the test function, and also started with the corrupted model shown in Table 3. This algorithm terminated on its own when it had converged within a minimum error value of $1.0e-11$, and after an elapsed time of 44 seconds. It converged to a final error value of $9.95e-12$. A comparison of the results in Table 5 with the known correct model from Table 2 shows that this algorithm has also converged to the known correct solution.

Units: mm & radians
 rms_error = 9.956050e-012

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0469	0.0008	-0.0133	4310.0793	0.0000
1	-0.0768	1.5708	0.0000	-350.0000	-684.7848
2	1.5708	3.1416	0.0000	1100.0000	0.0000
3	1.5708	1.5708	0.0000	-300.0000	0.0000
4	3.1416	1.5708	0.0000	0.0000	-1200.000
5	-3.1416	1.5708	0.0000	-0.0000	-0.0000
6	5.7633	1.5708	0.0000	50.8000	-184.1500

Table 5: Monte Carlo results on test function

4.1.4 GESA results on test function

The last algorithm to be run on the test function was the Guided Evolutionary Simulated Annealing algorithm. This too was started with the corrupted model shown in Table 3. This algorithm also terminated on its own when it had converged within a minimum error value of $1.0e-11$, after an elapsed time of 1 minute 44 seconds. It converged to a final error value of $9.99e-12$. Once again, a comparison of the results in Table 5 with the known correct model from Table 2 shows that this algorithm converged to the known correct solution.

Units: mm & radians
 rms_error = 9.985004e-012

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0469	0.0008	-0.0133	4310.0793	0.0000
1	-0.0768	1.5708	0.0000	-350.0000	-684.7848
2	1.5708	3.1416	-0.0000	1100.0000	0.0000
3	1.5708	1.5708	0.0000	-300.0000	0.0000
4	3.1416	1.5708	0.0000	0.0000	-1200.000
5	-3.1416	1.5708	0.0000	0.0000	-0.0000
6	5.7633	1.5708	0.0000	50.8000	-184.1500

Table 6: GESA results on test function

4.2 Simulated noise-free data set

Once the algorithms were proved to be working correctly, the next step was to apply them to ideal robot data. Ideal data was used to compare the algorithms in an environment free of unmodeled influences. Some examples of such influences in a real system are gravity droop, transmission backlash, and thermal expansion, to name a few.

A kinematic model using the nominal parameters presented in Table 2 was used to generate the simulated data. Using this model, 270 simulated data points were created by sweeping one joint of the simulated robot at a time from -45 degrees to $+45$ degrees of joint rotation. While each joint was swept, all the other joint angles were held constant at zero. Data points were generated for every two degrees of joint rotation, resulting in 45 points per joint. For each one of these points, the forward kinematics of this nominal model were computed. The result was a simulated set of CMM measurements of the robot's tool tip.

4.2.1 Steepest Gradient results on simulated noise-free data

The Steepest Gradient search algorithm was run on the simulated noise-free data set. This algorithm was started with the corrupted model presented in Table 3, which had an rms error of 7.7 mm with respect to the simulated noise-free data set. This algorithm was manually terminated after 35 minutes of execution. At that point, the algorithm had arrived at the parameter set shown in Table 7, which had an rms error of 0.07 mm with respect to the simulated noise-free data set.

Units: mm & radians
 rms_error = 7.180237e-002

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0468	0.0009	-0.0136	4310.0816	0.0000
1	-0.0767	1.5708	0.0000	-349.9991	-684.7862
2	1.5711	3.1416	-0.0000	1099.9991	0.0000
3	1.5706	1.5708	0.0000	-300.0020	0.0000
4	3.1415	1.5708	0.0000	0.0018	-1200.000
5	-3.1410	1.5708	0.0000	-0.0019	0.0000
6	5.7632	1.5708	0.0000	50.7983	-184.1504

Table 7: Steepest Gradient results on simulated noise-free data

4.2.2 Monte Carlo results on simulated noise-free data

The Monte Carlo algorithm was next applied to the simulated noise-free data set. Like the Steepest Gradient search, this algorithm was started with the corrupted model shown in Table 3. This search was also run for 35 minutes, at which point it was manually terminated. This search yielded the parameter set presented Table 8, which had an rms error of 0.07 mm with respect to the simulated noise-free data set.

Units: mm & radians
 rms_error = 7.305846e-002

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0468	0.0008	-0.0136	4310.2132	0.0000
1	-0.0757	1.5695	0.0000	-349.8442	-685.0096
2	1.5711	3.1405	0.0013	1100.2083	0.0000
3	1.5710	1.5699	0.0000	-299.7901	0.2249
4	3.1414	1.5712	0.0000	0.3044	-1200.081
5	-3.1412	1.5709	0.0000	0.3727	-0.0634
6	5.7642	1.5708	0.0000	50.6761	-184.1773

Table 8: Monte Carlo results on simulated noise-free data

4.2.3 GESA Results on simulated noise-free data

The next algorithm applied to the simulated noise-free data set was the Guided Evolutionary Simulated Annealing algorithm. Like the others, this search was initiated from the corrupted model shown in Table 3. This search was allowed to run until it was no longer making progress toward improving the rms error. The algorithm took 9 minutes to reach this point.

At that time, the algorithm was manually terminated. The resulting parameter set is shown in Table 9, which had an rms error of $5.61e-2$ mm with respect to this data set.

Units: mm & radians

rms_error = 5.614590e-002

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0468	0.0010	-0.0137	4310.3760	0.0000
1	-0.0768	1.5702	0.0000	-349.9895	-684.8611
2	1.5712	3.1417	0.0005	1099.9301	0.0000
3	1.5706	1.5706	0.0000	-300.2436	-0.2710
4	3.1421	1.5704	0.0000	0.1777	-1199.940
5	-3.1411	1.5707	0.0000	0.0198	-0.0351
6	5.7638	1.5708	0.0000	50.9051	-184.1089

Table 9: GESA results on simulated noise-free data

4.2.4 Circle-Point Results on simulated noise-free data

The last algorithm applied to the simulated noise-free data set was the Circle-Point algorithm described in section 3.5. This algorithm yielded the parameter set shown in Table 10, which had an rms error of $1.3e-5$ mm. This identification is nearly perfect, with errors attributable to round-off. The algorithm arrived at this solution after approximately 65 milliseconds of computation time. (As programmed, the complete algorithm was broken up into multiple programs that had to be run manually. It took approximately 4 minutes to run the whole algorithm.) This result verified that the algorithm was correctly implemented.

Units: mm & radians

rms_error = 1.298754e-005

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0469	0.0008	-0.0133	4310.0793	0.0000
1	-0.0768	1.5708	0.0000	-350.0000	-684.7848
2	1.5708	3.1416	0.0000	1100.0000	0.0000
3	1.5708	1.5708	0.0000	-300.0001	0.0000
4	-3.1416	1.5708	0.0000	-0.0000	-1199.999
5	-3.1416	1.5708	0.0000	-0.0000	-0.0000
6	5.7633	1.5708	0.0000	50.7999	-184.1500

Table 10: Circle-Point results on simulated noise-free data

4.3 Simulated data with noise

The next condition the algorithms were evaluated under consisted of a simulated data set similar to that used in section 4.2, but with noise added to the data. For this test, random noise between -1.0 and $+1.0$ mm was added to the Cartesian coordinates of the simulated data used in section 4.2.

4.3.1 Steepest Gradient results on simulated data with noise

As before, the first algorithm applied to the simulated data with noise was the Steepest Gradient algorithm. Once again, the algorithm was started with the corrupted model from Table 3, which had an rms error of 11.3 mm with respect to this data set. The algorithm ran for 35 minutes, after which time the algorithm was terminated manually. At this point, the algorithm yielded the parameter set shown in Table 11, which had an rms error of 0.99 mm with respect to this data set.

Units: mm & radians

rms_error = 9.932859e-001

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0468	0.0007	-0.0137	4310.0811	0.0000
1	-0.0768	1.5706	0.0000	-349.9995	-684.7868
2	1.5713	3.1416	-0.0000	1099.9986	0.0000
3	1.5707	1.5710	0.0000	-300.0022	0.0001
4	3.1417	1.5706	0.0000	0.0015	-1200.001
5	-3.1411	1.5710	0.0000	-0.0015	0.0005
6	5.7633	1.5708	0.0000	50.7982	-184.1512

Table 11: Steepest Gradient results on noisy simulated data

Since this data set had random noise on the order of 1 mm associated with it, an rms error result on the order of 1 mm was expected. This resulting kinematic model was then compared to the simulated noise-free data set. The purpose of running the four algorithms against the noisy data set was to see how well they could uncover the underlying model in the presence of random noise. To evaluate this, the resulting model was compared to the

simulated noise-free data set. This parameter set yielded an rms error of 0.15 mm, with respect to the simulated noise-free data set.

4.3.2 Monte Carlo results on simulated data with noise

The Monte Carlo algorithm was applied to the noisy simulated data in turn. As before, the algorithm was started with the corrupted model from Table 3. The algorithm was run for 35 minutes, after which time the algorithm was terminated manually. At this point, the algorithm yielded the parameter set shown in Table 12, which had an rms error of 0.99 mm with respect to the noisy simulated data.

Units: mm & radians

rms_error = 9.860493e-001

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0468	0.0006	-0.0139	4310.3434	0.0000
1	-0.0759	1.5698	0.0000	-350.0640	-685.0364
2	1.5713	3.1406	0.0009	1100.3238	0.0000
3	1.5710	1.5702	0.0000	-299.8870	0.2944
4	3.1425	1.5711	0.0000	0.0463	-1200.358
5	-3.1421	1.5714	0.0000	0.3147	-0.0008
6	5.7644	1.5708	0.0000	50.5663	-184.1310

Table 12: Monte Carlo results on noisy simulated data

This resulting kinematic model was then compared to the simulated noise-free data set. This parameter set yielded an rms error of 0.17 mm, with respect to the simulated noise-free data set.

4.3.3 GESA results on simulated data with noise

Like the others, the Guided Evolutionary Simulated Annealing algorithm was applied to the noisy simulated data, starting with the corrupted model from Table 3. After 4 minutes 30 seconds of execution time, the algorithm was terminated manually because it was no longer

making progress toward reducing the rms error. This algorithm yielded the model shown in Table 13, which had an rms error of 1.01 mm.

Units: mm & radians

rms_error = 1.009520e+000

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0467	0.0011	-0.0141	4310.2873	0.0000
1	-0.0772	1.5709	0.0000	-349.7289	-685.1955
2	1.5717	3.1418	-0.0002	1100.1769	0.0000
3	1.5703	1.5714	0.0000	-300.1892	-0.2123
4	3.1426	1.5714	0.0000	0.2215	-1199.740
5	-3.1401	1.5706	0.0000	0.1099	-0.3242
6	5.7640	1.5708	0.0000	50.5089	-184.1718

Table 13: GESA results on simulated noisy data

This resulting kinematic model was then compared to the simulated noise-free data set. This parameter set yielded an rms error of 0.22 mm, with respect to the simulated noise-free data set.

4.3.4 Circle-point results on simulated data with noise

Lastly, the Circle-Point algorithm was applied to the noisy simulated data. This identification technique yielded the parameter set shown in Table 14, which had an rms error of 1.13 mm with respect to the noisy simulated data set. The algorithm arrived at this solution after the same 65 milliseconds of execution time from before.

Units: mm & radians
 rms_error = 1.131778e+000

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0469	0.0006	-0.0140	4310.5273	0.0000
1	-0.0781	1.5721	0.0000	-349.1882	-683.8559
2	1.5721	3.1428	-0.0014	1100.2935	0.0000
3	1.5182	1.5422	0.0000	-372.5631	39.8033
4	3.1416	1.5467	0.0000	-9.0134	-1185.602
5	3.1381	1.5483	0.0000	-9.5450	-0.3108
6	5.7682	1.5708	0.0000	49.6178	-185.6149

Table 14: Circle-Point results on noisy simulated data

This resulting kinematic model was then compared to the simulated noise-free data set. This parameter set yielded an rms error of 0.58 mm, with respect to the simulated noise-free data set.

4.4 Calibration data set

The last set of data the algorithms were evaluated against was the calibration data set. This real world data was collected at Sandia National Laboratories in the following manner. The Cimatrix controller was programmed to send the P-8 robot to a sequence of positions in joint space. At each commanded pose, the robot was held for a settling time, then upon receiving a trigger (a manual carriage return, in this case) the joint sensor values were latched and recorded to disk. Through this sequence of robot positions, the SMX system tracked the mirror target at the robot's tool frame, and as the robot settled at each pose, the SMX controller was manually triggered to record the sensed Cartesian coordinates. The respective records of joint-space and Cartesian-space measurements were subsequently analyzed by the four algorithms presented here to calibrate the robot.

For this work, a particularly simple approach to data collection was taken. The data was collected by moving one joint at a time, while holding all the other joints fixed, resulting in a

data set with the desired properties. This data set consisted of 375 robot positions (80 for joint 1, 88 for joint 2, 89 for joint 3, 40 for joint 4, 38 for joint 5, and 40 for joint 6).

In addition to the calibration data set, a validation data set was also generated. This set consisted of 21 robot poses arbitrarily scattered throughout the robot's workspace. This data set was used to determine how well a calibrated model could explain data that was not included in the calibration process.

4.4.1 Steepest Gradient results on calibration data

The last data set the Steepest Gradient algorithm was applied to was the calibration data set.

This time, the search was started with the nominal (factory) model as shown in Table 2. The factory model had an rms error value of 4.77 mm with respect to the calibration data set.

This search algorithm yielded the parameter set shown in Table 15, which had an rms error of 1.69 mm, with respect to the calibration data set. The search was run for 35 minutes, and was terminated manually.

Units: mm & radians

rms_error = 1.692031e+000

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0467	0.0004	-0.0072	4310.0956	0.0000
1	-0.0770	1.5706	0.0000	-349.9824	-684.7709
2	1.5670	3.1412	0.0001	1100.0041	0.0000
3	1.5722	1.5713	0.0000	-299.9975	0.0003
4	3.1429	1.5722	0.0000	0.0291	-1199.987
5	-3.1347	1.5697	0.0000	-0.0169	-0.0076
6	5.7641	1.5708	0.0000	50.8084	-184.1285

Table 15: Steepest Gradient results on calibration data

This resulting kinematic model was then compared to the validation data set. This parameter set yielded an rms error of 5.25 mm, with respect to the validation data set. To better illustrate the difference between this result and the nominal factory model shown in Table 2,

Table 16 shows the Steepest Gradient result after subtracting the nominal factory model of Table 2 from it.

Units: mm & radians

Joint	Theta's	Alpha's	Beta's	a's	d's
0	-0.0002	-0.0004	0.0061	0.0163	0.0000
1	-0.0002	-0.0002	0.0000	0.0176	0.0139
2	-0.0038	-0.0004	0.0001	0.0041	0.0000
3	0.0014	0.0005	0.0000	0.0025	0.0003
4	0.0013	0.0014	0.0000	0.0291	0.0130
5	0.0069	-0.0011	0.0000	-0.0169	-0.0076
6	0.0008	0.0000	0.0000	0.0084	0.0215

Table 16: Nominal model subtracted from Steepest Gradient results on calibration data

4.4.2 Monte Carlo results on calibration data

The Monte Carlo algorithm was next run on the calibration data set. The search was started with the nominal (factory) model shown in Table 2. This algorithm ran for 35 minutes, at which time it was terminated manually. This algorithm yielded the kinematic model shown in Table 17, which had an rms error of 1.23 mm with respect to the calibration data set.

Units: mm & radians

rms_error = 1.229287e+000

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0469	0.0005	-0.0085	4309.7927	0.0000
1	-0.0769	1.5706	0.0000	-348.7656	-684.4297
2	1.5699	3.1415	0.0002	1100.1864	0.0000
3	1.5723	1.5708	0.0000	-300.6006	0.4708
4	3.1421	1.5698	0.0000	1.9968	-1200.543
5	-3.1381	1.5730	0.0000	1.3988	-1.2940
6	5.7630	1.5708	0.0000	50.6937	-182.4770

Table 17: Monte Carlo results on calibration data

This resulting model was then compared to the validation data set. It yielded an rms error of 3.95 mm with respect to the validation data. Table 18 shows the Monte Carlo result after subtracting the nominal model of Table 2 from it.

Units: mm & radians

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0000	-0.0003	0.0048	-0.2866	0.0000
1	-0.0001	-0.0002	0.0000	1.2344	0.3551
2	-0.0009	-0.0001	0.0002	0.1864	0.0000
3	0.0015	0.0000	0.0000	-0.6006	0.4708
4	0.0005	-0.0010	0.0000	1.9968	-0.5430
5	0.0035	0.0022	0.0000	1.3988	-1.2940
6	-0.0003	0.0000	0.0000	-0.1063	1.6730

Table 18: Nominal model subtracted from Monte Carlo results on calibration data

4.4.3 GESA results on calibration data

After the Monte Carlo algorithm was run on the calibration data set, the Guided Evolutionary Simulated Annealing algorithm was applied to this data set. Like the others, this algorithm was started with the nominal factory model presented in Table 2. This algorithm was terminated manually after 6 minutes of execution, when it was no longer making progress.

At that time, the algorithm had yielded the model shown in Table 19, which had an rms error of 1.44 mm, with respect to the calibration data set.

Units: mm & radians

rms_error = 1.441650e+000

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0468	0.0010	-0.0095	4310.9359	0.0000
1	-0.0772	1.5700	0.0000	-349.1852	-684.6061
2	1.5703	3.1418	-0.0002	1100.0199	0.0000
3	1.5720	1.5708	0.0000	-300.0530	0.3497
4	3.1428	1.5728	0.0000	0.7301	-1199.607
5	-3.1408	1.5703	0.0000	-0.4726	-0.1467
6	5.7625	1.5708	0.0000	50.7225	-183.5525

Table 19: GESA results on calibration data

This resulting model was then compared to the validation data set. It yielded an rms error of 3.26 mm with respect to the validation data. Table 20 shows the Guided Evolutionary Simulated Annealing result after subtracting the nominal model of Table 2 from it.

Units: mm & radians

Joint	Theta's	Alpha's	Beta's	a's	d's
0	-0.0001	0.0002	0.0038	0.8566	0.0000
1	-0.0004	-0.0008	0.0000	0.8148	0.1787
2	-0.0005	0.0002	-0.0002	0.0199	0.0000
3	0.0012	0.0000	0.0000	-0.0530	0.3497
4	0.0012	0.0020	0.0000	0.7301	0.3930
5	0.0008	-0.0005	0.0000	-0.4726	-0.1467
6	-0.0008	0.0000	0.0000	-0.0775	0.5975

Table 20: Nominal model subtracted from GESA results on calibration data

4.4.4 Circle-point results on calibration data

The last combination to be run was the Circle-Point algorithm against the calibration data set.

This technique yielded the kinematic model shown in Table 21, after 65 milliseconds of execution time. This model had an rms error of 3.31 mm with respect to the calibration data set.

Units: mm & radians

rms_error = 3.313578

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0469	0.0008	-0.0133	4310.0793	0.0000
1	-0.0768	1.5705	0.0000	-348.4788	-684.7848
2	1.5735	3.1414	0.0004	1101.4163	0.0000
3	1.5680	1.5729	0.0000	-302.9602	-2.3604
4	3.1400	1.5731	0.0000	0.1956	-1199.848
5	-3.1369	1.5711	0.0000	-0.4998	0.9638
6	5.7633	1.5708	0.0000	50.7301	-184.0987

Table 21: Circle-Point results on calibration data

This resulting kinematic model was then compared to the validation data set. This parameter set yielded an rms error of 2.52 mm, with respect to the validation data set. Table 22 shows the Circle Point result after subtracting the nominal model of Table 2 from it.

Units: mm & radians

Joint	Theta's	Alpha's	Beta's	a's	d's
0	0.0000	0.0000	0.0000	0.0000	0.0000
1	0.0000	-0.0003	0.0000	1.5212	0.0000
2	0.0027	-0.0002	0.0004	1.4163	0.0000
3	-0.0028	0.0021	0.0000	-2.9602	-2.3604
4	-0.0016	0.0023	0.0000	0.1956	0.1520
5	0.0047	0.0003	0.0000	-0.4998	0.9638
6	0.0000	0.0000	0.0000	-0.0699	0.0513

Table 22: Nominal model subtracted from Circle Point results on calibration data

5.0 CONCLUSIONS

The work presented here provides a comparison of four different methodologies to the problem of robot calibration. The four different methodologies consist of a gradient based iterative approach, two evolutionary approaches, and a non-iterative approach. In reviewing the results presented in section 4.0, it can be seen that overall, the best methodology is the non-iterative Circle-Point approach.

First, the search results presented in section 4.2 associated with the simulated noise-free data set are considered. On this data set, the Circle-Point method is seen to be clearly superior. The Circle-Point method had an rms error of $1.30e-5$ mm, while the Steepest Gradient, Monte Carlo, and Guided Evolutionary Simulated Annealing methods had rms errors of $7.18e-2$ mm, $7.3e-2$ mm, and $5.61e-2$ mm, respectively. The Circle-Point method yielded an rms error value that was three orders of magnitude better than any of the other search techniques tried. Furthermore, the Circle-Point method arrived at its solution significantly faster than any of the other identification methods. It found its solution in only 65 milliseconds, while the next fastest method took 9 minutes.

Next, the results of section 4.3 are considered. When the four techniques were run on the noisy simulated data set, they all ended up with an rms error value very close to 1.0 mm. The Circle-Point method was the worst of all, with an rms error of 1.132 mm. The Steepest Gradient, Monte Carlo, and Guided Evolutionary Simulated Annealing methods had rms error values of 0.993 mm, 0.986 mm, and 1.010 mm, respectively. To see how well the four models did at uncovering the underlying model, these results were then compared to the

simulated noise-free data set. On this data set, the Steepest Gradient, Monte Carlo, GESA, and Circle Point algorithms had rms error values of 0.15 mm, 0.17 mm, 0.22 mm, and 0.58 mm, respectively. This was somewhat surprising because it was thought that the Circle-Point method would be less sensitive to random noise, since it does not try to follow the terrain of the function. Although the result of the Circle-Point method is slightly worse than the others, it is still considered the superior method on this data set. This is because once again, the Circle-Point method arrived at its solution significantly faster than the other methods. The Circle-Point method arrived at its solution in 65 milliseconds, while the next fastest technique, the GESA algorithm, took 4.5 minutes.

The last data set to be considered is the Calibration data set of section 4.4. On the calibration data set, once again, the Circle-Point method's result was slightly worse than the other three methods. The Circle-Point method had an rms error of 3.3 mm, while the Steepest Gradient, Monte Carlo, and Guided Evolutionary Simulated Annealing methods had rms errors of 1.69 mm, 1.22 mm, and 1.44 mm, respectively.

It is important to note, however, that when these three kinematic models were evaluated against the validation data set (which was not used for calibration), the model resulting from the Circle-Point method had the lowest rms error. On the validation data, the Circle-Point model had an rms error of 2.52mm, while the Steepest Gradient, Monte Carlo, and Guided Evolutionary Simulated Annealing methods had rms errors of 5.25 mm, 3.95 mm, and 3.26 mm, respectively. This indicates that while the models generated by the others do a better job of explaining the calibration data set, the Circle-Point model seems to do a better job of describing the robot system in general. This is, after all, the goal of the calibration process.

In addition to having the lowest rms error for the validation data, the Circle-Point method once again arrived at its solution much faster than any of the other methods. As before, the Circle-Point method took 65 milliseconds to arrive at its solution, while the next fastest method, the GESA algorithm, took 6 minutes. For these two reasons, the Circle-Point algorithm is again considered superior.

There is one other factor to be considered. The gradient search and the two evolutionary searches all require some initial starting point. In most cases, the nominal robot parameters are known and can be used as a good initial guess. In some cases, however, this information might not be known, requiring the algorithm to be started from some randomly-generated initial parameter set. How close this starting point is to the correct solution will affect how long the search takes to converge, and may prevent the search from converging altogether. The Circle-Point method, on the other hand, does not require a starting point. This is an advantage because it means that the time it takes to go through the identification process should be a constant, regardless of the data set used or the accuracy of an initial model.

There is, however, one disadvantage to the Circle-Point method that is worth mentioning. Unlike the other techniques presented, the Circle-Point method requires that the calibration data points be positioned in circular arcs about each joint axis. The other techniques do not have this requirement, although the placement of the calibration points can clearly have an impact on the successfulness of the calibration process.

In summary, this work shows that the Circle-Point identification algorithm is a highly attractive technique. Its use is recommended wherever the data acquisition process satisfies the requirements.

REFERENCES

1. Newman W. S., *et al.*, *Calibration of a Motoman P8 Based on Laser Tracking*. Center for Automated Intelligent Systems Research, Technical Note TR 98-105. 1998: Case Western Reserve University, OH.
2. Nyberg, M. and Pao, Y.H., *Automatic Optimal Design of Fuzzy Systems Based On Universal Approximation and Evolutionary Programming*. Logic and Intelligent Systems, H. Li and M.M. Gupta (Eds.) 1995: Kluwer Academic Publishers, MA. pp. 311-366.
3. Mooring, B. W., *et al.*, *Fundamentals of Manipulator Calibration*. 1991: John Wiley & Sons, NY. p. 135-140.
4. Stone, H. W., *Kinematic Modeling, Identification, and Control of Robotic Manipulators*. 1986: Kluwer Academic Publishers, MA. p. 47-64.
5. Mikhailov, G. A., *Minimization of Computational Costs of Non-Analogue Monte Carlo Methods*. 1991: World Scientific, NJ. p. 133-135.
6. Hollerbach, J.M., *The Calibration Index and Taxonomy for Robot Kinematic Calibration Methods*. International Journal of Robotics Research, 1996. 15(12): p. 573-591.
7. Ikits, M. and J.M. Hollerbach. *Kinematic Calibration Using a Plane Constraint*. in *Proceedings of International Conference on Robotics and Automation*. 1997: IEEE. p. 3191-3196.
8. Osborn, D.W. and W.S. Newman. *A New Method for Kinematic Parameter Calibration via Laser Line*. in *Proceedings of International Conference on Robotics and Automation*. 1993: IEEE. Vol. 2, p. 160-165.
9. Judd, R.P. and A.B. Knasinski, *A Technique to Calibrate Industrial Robots with Experimental Verification*. IEEE Transactions on Robotics and Automation, 1990. 6(1): p. 20-30.
10. Anderson, R.L., *et al.*, *Open-architecture Controller Solution for Custom Machine Systems*. in *SPIE*. 1996.
11. Denavit, J. and R.S. Hartenberg, *A kinematic notation for lower-pair mechanisms based on matrices*. ASME Journal of Applied Mechanics, 1955: p. 215-221.
12. Hayati, S.A., *Robotic arm geometric parameter estimation*. in *22nd IEEE International Conference on Decision and Control*. 1983: IEEE. Vol. 3 p.1477-1483
13. Wolovich, *Robotics: Basic Analysis and Design*. 1987: Holt, Rinehart and Winston, NY p. 62-91.
14. Asada, H. and J.-J.E. Slotine, *Robot Analysis and Control*. 1986: John Wiley and Sons, NY. p. 31-34.
15. Craig, J.J., *Introduction to Robotics*. 1986: Addison-Wesley, MA. p. 60-75.
16. Fishman, G.S., *Monte Carlo Concepts, Algorithms, and Applications*. 1996: Springer-Verlag, NY. pp. 335-388.