International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:7, No:9, 2013

Distributed Motion Control Real-Time Contouring Algorithm Implementation and Performance Test

Francisco J. Lopez-Jaquez, Sandra E. Ramirez-Jara

***Abstract*—**This paper presents an implementation and performance test of a distributed motion control system based on a master-slave configuration used to move a plasma-cutting torch over a predefined trajectory. The master is a general-purpose computer running on an open source operating system platform and software developer. Software running in the master computer generates commands on real time and we measure performance based on a selected set of differences between expected and observed distances. We are testing the null hypothesis that the outcome trajectory is identical to the input against the alternative hypothesis that there is a shift to the right or left of the input one. We used the Wilcoxon signed ranks test method for the hypothesis test.

***Keywords*—**Distributed, motion, control, real-time, contouring.

I. Introduction

DURING development of a plasma-cutting X-Y table, it was required to probe if the distributed motion control system (DMCS) was able to reproduce the trajectories as required. This paper presents the results of a test of the motion control system including details of the implementation in this section, and the network and algorithm sections. DMCS is one of the most demanding applications in terms of real-time [1]-[3]. We present the details of the DMCS performance test in the experiment and results sections and paper finish with concluding and further research remarks.

The cutting system is able to produce a contouring operation on a plane through a set of stepper motors that move a cutting torch. The master computer generates the DMCS commands running a piece of software that we have developed. This software runs under Linux Ubuntu operative system and it does all the required operations for the DMCS. The software is open and we can modify it to include other components from different manufacturers and provides the interface to accomplish the tasks required by the DMCS. Nowadays openness is the focus of DMCS systems [4]-[6]. The software reads a set of control points from a file during loading, sets the network parameters, displays the trajectory control points and waits for input to start the trajectory traversing, among other initialization tasks.

Fig. 1 shows the basic elements that are part of the DMCS. An RS-485 network links the master (a general-purpose computer) and the motor drivers (the slaves). The stepper motors, connected to their associated motor driver, are in charge of turning a power screw to convert turning motion into linear movement fixing the cutting torch holder to a screw nut. A set of two parallel bars keep the cutting torch perpendicular to the working piece. The cutting table holds a metal sheet in place and the plasma torch moves over it, on an XY plane. The computer issues motion control commands to the motor drivers at eight milliseconds, sampling time. The commands include speed and direction at which the motors must turn, the motor drivers get the commands and turn the motors in the specified direction and speed. The input to the motion control is a list of control vertices, time stamps and flags to signal the start and end of the trajectories.

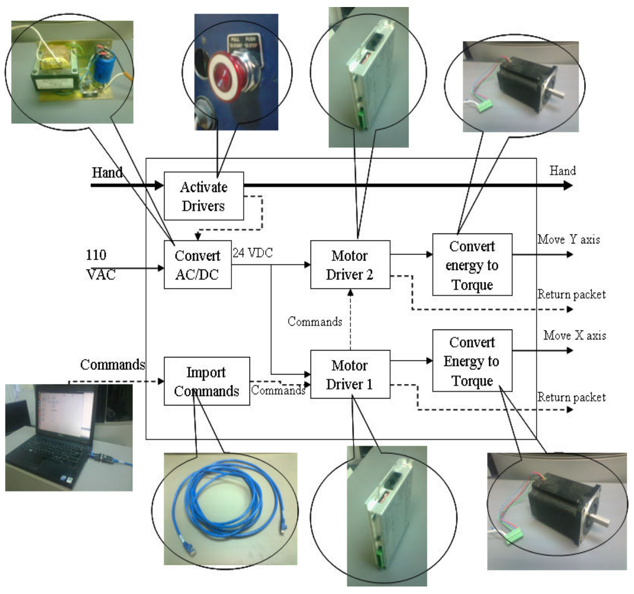

Fig. 1 Coordinate motion control elements

Once a trajectory traversing starts a timer is set, sampling time, speed and direction are calculated base on time increment and current position. Position is calculated applying interpolation according to the input control points.

The stepper motors are NEMA 34, holding torque 1,282 oz-in, 1.4 amps per phase, 11 VDC rated voltage with step angle of 1.8 degrees per full step. Their axes are direct interconnected to a spline screw of 38.1mm of pitch, i.e., one turn of the screw results in 38.1mm displacement of the cutting torch.

The motor drivers are model LS-146 intelligent micro stepping controller with integrated amplifier for applications using two-phase stepper motors, 32-bit position counter, 1/8,

F. J. is with the Industrial and Manufacturing Engineering, Universidad Autonoma de Ciudad Juarez, Chihuahua, Mexico (corresponding author; phone: 52-656-6884843; e-mail: frlopez@ uacj.mx).
S. R. is with the Electrical and Computer Engineering Department, Universidad Autonoma de Ciudad Juarez, (e-mail: sramirez@luacj.mx).

1/4, 1/2 micro step and full step capable. Position and velocity modes, communication speed of 19.2 to 115.2 Kbps, among other features [7].

The computer is a laptop with Core 2 Duo CPU T7250 @ 2.00 GHzx2, 32-bit OS and 3.9GiB of RAM, Ubuntu 12.04 LTS operating system. The motion control software was developed using Qt Creator 2.3.

II. The Network

An RS-485 network interconnects the motion control elements [8]. In this case, the master computer does not have an RS-485. An USB to RS-232 and a RS-232 to RS-485 adapters connects the master to the network. During initialization of the motor drivers, they require a network communication speed of 19.2Kbps but, once the initialization ends, speed is set to higher speed for a successfully implementation [5]. In this case, network communication speed was set to 38.4Kbps but it can be set up to 112.4Kbps that is the limit on the motor drivers.

We cannot achieve a target sampling time of eight milliseconds at 19.2Kbps, for a sampling rate of 125 Hertz. Bites will accumulate in the communication buffer at 19.2Kbps, resulting on sampling time instability. At 19.2Kbps it is possible to send 153 bits each eight milliseconds, assuming that the sampling time interval is stable, but if we need to send more data, it will start to accumulate in the transmission buffer producing a delay on the communication [9]. Every byte that is send requires start and end bites; to send a byte it requires 10 bits then it is plausible to send 15 bytes every sampling time interval, neglecting any other operating system activities. During command generation, every eight milliseconds, theoretically, the master sends data to the two motor drivers. The load trajectory command consists of seven bytes, for each motor, and if a change of direction is required four bytes to stop and another four bytes to start the motor. This clearly goes over the 153 bits limit imposed by the eight milliseconds sampling time at 19.2Kbps

The transmission buffer of the port will accumulate the information; the cutting torch will be behind in relation to the position in the computer screen, also the sampling intervals will show instability, hanging the timer more than expected. Fig. 2 shows a chart of sampling time for 19.2Kbps and 38.4Kbps and it can be observed that at 38.4Kbs the sampling time is stable while at 19.2KBps sampling time fluctuates up to 80 milliseconds in a constant pattern leading to a misplaced trajectory.

The motor drivers initialization require to send commands to assign the network address to each one of the motor drivers, this task is done writing a string to the communication port, i.e., to assign address 1 to first motor driver in the line string of characters {0xAA, 0x00, 0x21, 0x01, 0xFF, 0x00+0x21+0x01+0xFF}. All commands use 0xAA to signal the beginning of the packet, a command. At start up, default address is 0x00 for the first motor driver on the network line. 0x21 is the command identifier where first digit, 2, indicates the number of data bytes that follows. The second digit, 1, identifies the address assignation command. 0x01 is the new address that is going to be assigned to first motor driver in the network line. 0xFF is a group address assignation, the master send commands to this address when all motor drivers in the group must process the command. Finally, 0x00+0x21+0x00+0xFF is the check sum byte used by the motor driver to validate the packet command.

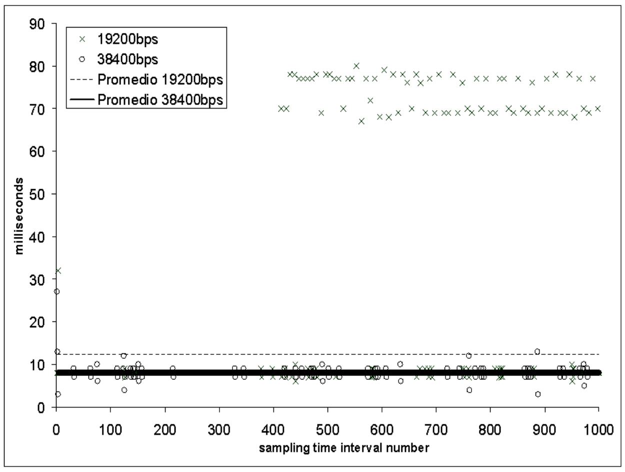

Fig. 2 Sampling time at 19.2 and 38.4 Kbps

Once the software initializes the motor drivers, the next task is to initialize their operating parameters. The initialization requires specification of the speed factor, active or deactivate limit switch auto stop, minimum profile velocity, holding and running current that the motor driver must deliver (accordingly to the motor specifications). In this case, the packet command for first motor driver is {0xAA, 0x01, 0x56, 0x07, 0x01, 0x32, 0x19, 0x00 0x01+0x56+ 0x07+0x01+ 0x32+0x19+0x0}. First two bytes are the beginning of packet identification and address of the motor driver that must process the packet. 0x56 is the command byte where first digit 5 indicates that there are 5 bytes of data following this byte and second digit, 6, is the command identification. 0x07 is the control byte where first two bits are used to specify the speed factor; third bit is used to specify if the limit switch auto stop must be disabled or enabled. Forth bit indicates if motor must turn off on limit switch. Fifth bite indicates if motor must turn off on stop switch. The motor driver does not use the last three bites of this control byte. In this case this byte is set to 0x07 that is "00000111" which means that sped factor is set to 1 and limit switch auto stop is disabled, currently the machine does not have any limit switch attached to the motor driver and it requires to be set to ignore it in order to activate the motor. 0x01 is the minimum profile velocity, running current is set to 50 and holding current to 25 (0x32 and 0x19, respectively) to deliver 1.0 Ampere to the stepper motor on normal running and up to 1.4 Amperes pick current. 0x00 is to indicate that the thermal limit is disabled. Finally, the check sum byte that is the sum of address byte up to the thermal limit set byte.

III. The Motion Control Algorithm

Once a trajectory traversing start button is clicked on the display interface, it get current time in milliseconds T1 from the systems clock, set initial position X and Y to the first control point of the interpolation window, traversing time (Tt) is set to zero, sets the timer to 10 milliseconds. The CGMCS is called after 10 milliseconds, current time in milliseconds is again obtained and saved as T2, elapsed time is calculated from T2 and T1 and stored as T0, this is the time increment. The algorithm seta Tt to previous Tt + T0 and it is used to compute the interpolation X and Y point associated to it, this is the current position.

In the pseudo code list, when the software actives a new interpolation windows it computes the Lagrange polynomial interpolation denominator. This is done as listed from 1 to 10 in the pseudo code, e.g. at the beginning of the traversing the interpolation window is set to 1, and assuming linear interpolation, the windows will contain two X,Y,T triplets, one signaling the beginning of the trajectory and the other the next control point and its associated time stamp. Let's call them Wx[1][2]={180,182}, Wy[1][2]={38,40} and Wt[1][2]={0,2}. The Tt will start a Wt[1][1]+sampling time and while Tt is lower than Wt[1][2] the interpolation points are computed using data from this window and because the Lagrange polynomial keeps the same while the interpolation control points are the same it is only calculated once, at the beginning of the every interpolation window. The code from 11 to 24 runs every time, at the sampling time rate, because it changes for every value for Tt, the accumulated traversing time.

The input data is a set of k interpolation windows Wx[k][nP]={set of X data values associated to the control points},Wy[k][nP]={a set of Y data values associated to the control points} and Wt[k][nP]={stamp times associated to the control points}

```
1:   if interpolation window changes then
2:    for i equals to 0
3:      Wd[i] equals to 1
4:        for j equals to 0
5:          if j is different than i
6:            increase Wd[i] by (Wd[i]*(Wt[k][i]-Wt[k][j]))
7:          increase j by one
8:        while j less than nP repeat from line 4
9:        increase i by one
10:       while i less than nP repeat from line 2
11:  else
12:       Set X and Y to zero
13:       for i equals to 0
14:          set mL equals to 1
15:          for j equals to 0
16:            if j is different than i
17:              incress mL by (Tt/1000 –Wt[k][j])
18:            increase j by one
19:          while j less than nP repeat from 14
20:         increase mL by (mL / Wd[i])
21:         increase X by (mL*Wx[k][i])
22:         increase Y by (mL * Wy[k][i])
23:          increase i by one
24:       while i less than nP repeat from 12
```

It can be observed that the computation of the Lagrange polynomial for a second order or higher does not require any change in the algorithm, just by increasing the nP value it will be able to generate higher order interpolation, i.e. a value of nP = 2 will generate second order interpolation points giving a smooth trajectory if it is desirable. A higher value of nP will require more time to calculate the Lagrange polynomials. In this case linear interpolation was used, nP=1.

Fig. 3 shows the flow diagram for the commands generation of the motion control system (CGMCS). We use Lagrange polynomial interpolation to generate the (X,Y) points.

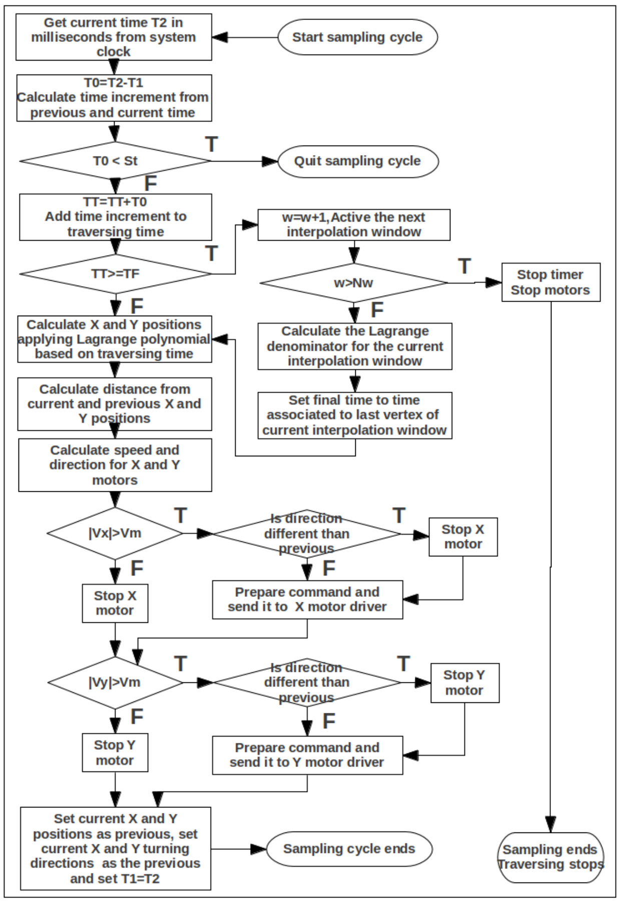
Get current time T2 in milliseconds from system clock
Start sampling cycle
T0=T2-T1 Calculate time increment from previous and current time
T0 < St
T
Quit sampling cycle
F
TT=TT+T0 Add time increment to traversing time
w=w+1,Active the next interpolation window
TT>=TF
T
w>Nw
T
Stop timer Stop motors
F
F
Calculate X and Y positions applying Lagrange polynomial based on traversing time
Calculate the Lagrange denominator for the current interpolation window
Calculate distance from current and previous X and Y positions
Set final time to time associated to last vertex of current interpolation window
Calculate speed and direction for X and Y motors
|Vx|>Vm
T
Is direction different than previous
T
Stop X motor
F
F
Stop X motor
Prepare command and send it to X motor driver
|Vy|>Vm
T
Is direction different than previous
T
Stop Y motor
F
F
Stop Y motor
Prepare command and send it to Y motor driver
Set current X and Y positions as previous, set current X and Y turning directions as the previous and set T1=T2
Sampling cycle ends
Sampling ends Traversing stops

Fig. 3 Command generation flow diagram

The software computes the displacements from current and previous X-Y positions and then computes the required speed and direction for each displacement. Using the current cutting torch speed the software computes the component speed X and Y. The signs of the displacements define the direction of the movements; a negative displacement means a counterclockwise rotation of the motor shaft and clockwise rotation if the displacement is positive. The software issues a load trajectory command to each motor driver based on the computed speed and direction.

If the component speed does not reaches a minimum of 10 steps per second the velocity is set to zero and a stop turning command is issue to the associated motor driver. Even more, when a change in direction occurs, the software first send a motor stop command before the load trajectory command, and initiate movement command after this. The command structure follows the structure defined by the motor driver manufacturer, and in general the standard used for RS-485 network [7], [8].

The motor drivers are capable of operating in velocity, position (trapezoidal) and unprofiled modes. The one used in this case is unprofiled motion that requires setting of a timer in the motor driver to an initial timer count (ITC). Equation (1) is used to determine the ITC for a required unprofiled speed, S, in steps per second. The constant values 65538 and 625000 are specific for this kind of motor drivers and the speed factor at which it is set to operate [1].

The load trajectory command contains the information to perform the unprofiled movement and it consist of a string of characters, packet. The string consist of character 0xAA to signal start of packet, 0x01 or 0x02 to signal the address of the motor driver that should process the command, 0x34 that identifies command load trajectory and includes the number of bytes that came after it, one control byte and two bytes to indicate the speed. Finally, a check sum byte that consists of adding address, command, control and the speed data bytes. The tricky part here is the preparation of the speed data bytes that need to be send least significant byte first. After component speed in steps per second (S) is converted to ITC using (1). Using (2) and (3) is possible to split the resulting ITC value into two bytes (SB1 and SB2), second byte is placed first in the command string.

$$ITC = 65538 - (625000 / S) \tag{1}$$
$$SB1 = ITC / 256 \tag{2}$$
$$SB2 = ITC - 256(SB1) \tag{3}$$

During trajectory traversing, it is possible to hang the movement without disturbing the trajectory traversing or sampling time interval, since the algorithm works with time increments, the CGMCS timer can be stop and continue at any time during the trajectory traversing. The software includes a button to get the stop/continue signal from the user. The continue signal starts again the CGMCS timer and time increment is computed setting the previous time as the one when the user clicked the continue button. This adds flexibility to stop and continue at any time providing a button on the display interface to click on.

The CGMCS algorithm, network initialization, parameter setting, and control points trajectories loading and editing where included in a software package where the operator is able to simulate trajectory traversing, edit control points, assign stop flags, move the cutting torch on the XY plane, reset network, change network communication speed, change traversing speed, among other options are included in the software. Fig. 4 shows a screen shoot of the software main screen showing a set of buttons to move the cutting torch and the trajectory traversing operation.

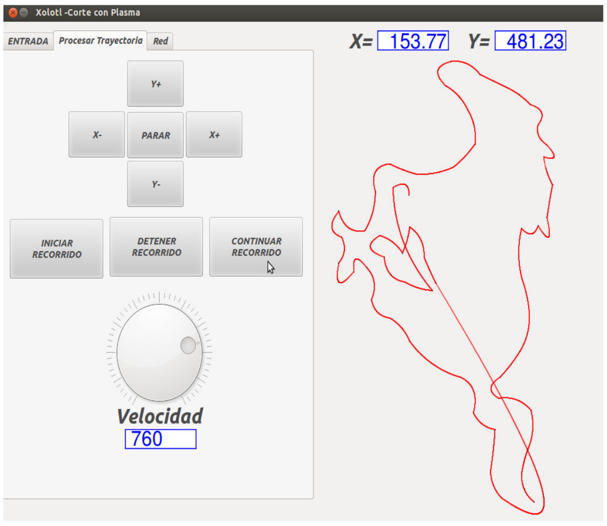

Fig. 4 Screen shoot of the motion control software

IV. The Experiment

Fig. 5 shows the test specimen, a trajectory consisting of 103 control points. The control points are not equidistant along the trajectory. The algorithm generates X-Y points applying linear interpolation based on elapsed time, position is a function of time. The sampling time is set to 8 milliseconds, resulting torch velocity is set to 760 steps per second and component X and Y velocities are calculated proportional to the required X and Y position displacements. Network communication speed is set to 38.4 Kbps during trajectory traversing.

The controls points include a stop flag and the software is able to stop the trajectory traversing at that point and then continue after user hits the continue button on the computer interface. Each time the cutting torch reaches a stop control point the software actives the plasma to make a piercing operation, triggering the plasma generator. Fig. 5 shows the stop points along the other control points of the trajectory. Traversing of the trajectory starts at point 1 and will stop at points 2 to 14 as they appear during trajectory traversing.

In this experiment, we are finding data to probe whether the cutting torch traverses the test trajectory accordingly to the input control points. If this is the case, the differences between the measured and calculated distances should be equal to cero. Five specimens where run to estimate the average measured distances.

We considered thirty different distances in this experiment, and five measurements for each distance as listed on Table I. Column A lists the reference points of the measured distance, i.e. from first row, distance 14 is from point 1 to 9, distance 26 is from point 7 to 10, and so on. Column B shows the theoretical distance and column C lists the average distance calculated from five observed measurement. D and |D| are the differences obtained from C-B and the absolute value, respectively. Column E is the ranking, F id the sign and last two columns are the positive and negative T values, totals for each one are included in the bottom row. We took the measurements rounded to millimeters based on appreciation and from there we calculate the average. We used the physical system illustrated in Fig. 6.

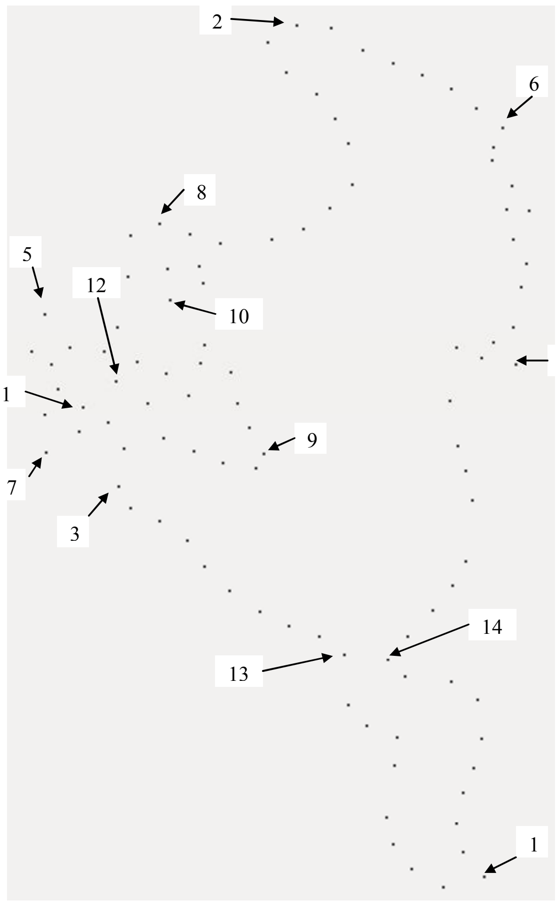

Fig. 5 Test specimen input control points

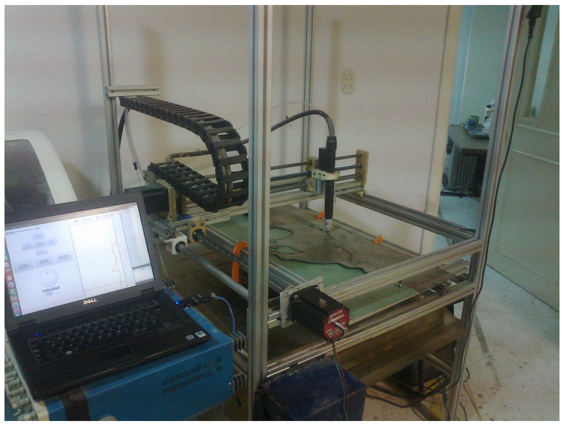

Fig. 6 The physical system

V. The Analysis

The null hypothesis is that the difference between the outcome measurements and the theoretical distances are identical against the alternative hypothesis that there is a shift to the right or left of the other, there is a significant difference between them.

We used the Wilcoxon T statistic, signed rank test, for matched-pairs design test. We have selected this test because we are comparing two probability distributions where we have obtained measured and theoretical data from the same distances and we do not know if the data comes from a normal distribution at all. Wilcoxon test is part of a distribution free test of the non-parametric methods that focus on the location of the probability distribution rather than on specific parameters of the population [10]. On Table I, there is the data outcome from the analysis. Distances appear sorted in ascending order, based on the absolute differences, column |D|.

The T value is 182 and it corresponds to the negative signed rank differences. According to the procedure, T is the lower absolute value of the positive or negative differences. According to the test procedure, we must reject the null hypothesis if the T vale, calculated from the observed data, is lower or equal to the T_0, the test value obtained from the critical values in the Wilcoxon matched-pairs signed rank test chart, [10]. In this case, for n = 30, level of significance equal to 5% and two tail test, $T_0 = 137$. Therefore, we cannot reject the null hypothesis. The evidence is not strong enough to indicate a shift in the distributions of ratings for the distances.

VI. Conclusions

We have put the distributed motion control system, as implemented, under evaluation. Neglecting mechanical issues or any other factors not considered here, we have compared the expected and real trajectories based on a set of predefined distances among the trajectory; the ability to reproduce the trajectory was the main characteristic under evaluation. Based on the outcome of the experiment, we can say that if any other factors, not considered in the test, do not disturb the runs the resulting trajectory is identical to the expected one.

The central part of the motion control system is a piece of software that generates motion control commands on real-time, at eight milliseconds sampling rate. If the operating system is able to maintain this sampling rate and there are not communication pitfalls in the transmission-reception of data, the algorithm produces good results according to the statistical analysis.

Currently the algorithm handles feedback from the motor drivers' current position but if the sampling rate takes too long the motors will continue to turn during that period and probably will produce a shift on the trajectory. When a command does not reach the motor driver due to problems somewhere in the communication line this will probably introduces another shift in the trajectory.

The runs in the experiment where all runs that do not observed any of the two problems mentioned before. We are going to conduct more research to attack these problems and make the algorithm robust to these issues.

TABLE I
DISTANCES RATINGS FOR WILCOXON'S SIGNED RANK TEST

A	B	C	D	|D|	E	F	T+	T-
14(1-9)	367.18	367.0	0.18	0.18	1	1	1	0
26(7-10)	150.86	150.6	0.26	0.26	2	1	2	0
13(1-8)	561.73	562.0	-0.27	0.27	3	-1	0	3
16(1-11)	473.48	473.2	0.28	0.28	4	1	4	0
15(1-10)	505.53	505.2	0.33	0.33	5	1	5	0
18(1-13)	202.04	202.4	-0.36	0.36	6	-1	0	6
19(1-14)	183.17	182.6	0.57	0.57	7	1	7	0
5(9-10)	138.57	139.2	-0.63	0.63	8	-1	0	8
30(7-14)	304.44	303.8	0.64	0.64	9	1	9	0
17(1-12)	473.85	475.0	-1.15	1.15	10	-1	0	10
4(7-8)	196.79	195.4	1.39	1.39	11	1	11	0
8(1-3)	409.80	411.2	-1.40	1.40	12	-1	0	12
1(1-2)	674.13	675.6	-1.47	1.47	13	-1	0	13
12(1-7)	466.70	465.2	1.50	1.50	14	1	14	0
11(1-6)	579.17	577.6	1.57	1.57	15	1	15	0
23(7-5)	107.00	105.2	1.80	1.80	16	1	16	0
29(7-13)	274.61	277.0	-2.39	2.39	17	-1	0	17
27(7-11)	44.82	41.8	3.02	3.02	18	1	18	0
10(1-5)	547.83	544.8	3.03	3.03	19	1	19	0
20(7-2)	381.66	378.6	3.06	3.06	20	1	20	0
21(7-3)	60.84	64.0	-3.16	3.16	21	-1	0	21
25(7-9)	165.00	168.4	-3.40	3.40	22	-1	0	22
28(7-12)	76.38	80.4	-4.02	4.02	23	-1	0	23
9(1-4)	396.73	401.0	-4.27	4.27	24	-1	0	24
7(13-14)	33.24	28.4	4.84	4.84	25	1	25	0
6(11-12)	32.02	37.6	-5.58	5.58	26	-1	0	26
22(7-4)	362.44	354.8	7.64	7.64	27	1	27	0
3(5-6)	375.69	366.8	8.89	8.89	28	1	28	0
2(3-4)	315.34	304.6	10.74	10.74	29	1	29	0
24(7-6)	427.45	416.0	11.45	11.45	30	1	30	0
						T	268	182

Currently, if the operating system hangs more than expected the software display a discontinue trajectory instead of a solid one when there are not operating systems hangs during the trajectory traversing operation.

There is also place to experiment the behavior at different end velocities, movement strategies and high order interpolation, among others. The base has been set to keep improving the algorithm and, at the end, take the complete motion control system and evolve it into a better one, but we need to conduct more research.

REFERENCES

[1] C. Pang, V. Vyatkin and C. Fantuzzi, "time-complemented event-driven control framework for distributed motion control systems based on IEC 61499 and IEEE 1588," 9th International conference on Industrial Informatics (INDIN), 2011, pp. 640-645

[2] G. Y. Gu, L. M. Zhu, Z. H. Xiong and H. Ding, "design of a distributed multiaxis motion control system using the IEEE- 1394 bus," IEEE Transactions on Industrial Electronics, vol. 50, no. 2, December 2010, pp. 4209-4218

[3] Z. Wang, D. Yu, Y. Hu and Y. Tao, "design of distributed cross-coupled controller based on motion control bus," IEEE 2009 chinese control and decision conference (CCDC 2009), pp. 1662-1667

[4] Shi Hongyu, Feng Yong, and Chen Na, "a distributed digital motion control system based SERCOS," IEEE international conference on electrical and control engineering, 2010, 48-52

[5] F. Benzi, G. S. Buja and M. Felser, "Comunication architecture for electric drives," IEEE Transactions on industrial informatics, Feb. 2005, vol. 1, pp. 47-53.

[6] S.Y. Lin, C. H. Ho and Y. Y. Tzou, "distributed motion control using real-time network comunication techniques," power Electronics and motion control conference, IEEE IPMEC 2000, pp. 843-847

[7] Logosol Intelligent Microstepping driver LS-146, http://www.logosolinc.com/products/ls-146.htm

[8] J. Axelson, "An RS-485Network— in Serial Port Complete," second edition, Lakeview Research, Madison WI, 2007, pp. 281-31

[9] M. Aqil, N. Masud, S. M. Pasha and S. Nazir, "a novel control scheme for real time motion control," 9th International Multitopic Conference, IEEE INMIC 2005, pp. 1-5

[10] M. William and S. Terry, "Nonparametric statistics— in Statistics for Engineering and the Sciences," Fifth Edition, editorial Prentice Hall, Upper Saddle River, New Jersey, 2007, pp 755-778.