A new method for motion synchronization among multivendor's programmable controllers

Jacopo Cavalaglio Camargo Molano, Achraf Lahrache, Riccardo Rubini, Marco Cocconcelli

Department of Sciences and Methods of Engineering, University of Modena and Reggio Emilia, Reggio Emilia, Italy

Abstract

This paper is aimed at increasing the number of possible architectures of distributed control systems by investigating and developing novel methods for the synchronization of axes between PLCs and iPCs of different vendors. In order to find a global solution to this problem, particular attention has been focused on programmable controllers that can manage axes by means of point-by-point control or motion instructions.
Two synchronization algorithms have been developed and validated for real and virtual axes; they differ in computational load so that they can be used with programmable controllers having high or low computational performances.

Keywords: Synchronization, Coordinated motion control, Real-Time Ethernet network, Multi-axis motion

1. Introduction

Distributed solutions for real-time system architectures are more and more used for the improvement of the industrial process and the development of smart factories in order to follow the new wave of Industry 4.0 [1]. In comparison with the centralized architecture, these systems improve the dependability, compensability, scalability and extensibility of the products [2]. They are also called networked motion control systems (NMCSs) and they consist of a set of different nodes such as controllers, sensors, drive controllers, regulators, HMIs and actuators, spatially distributed and interconnected by a communication network [3, 4].

In the automation field [5] there are several communication networks and communication protocols that answer different requirements for different applications. Real-time Ethernet (RTE), which is a field bus technology, is commonly used for the communication of data among nodes of distributed real-time systems thanks to its security and reliability [6, 7, 8]. Several Ethernet applications exist such as ControlNet [9], PROFIBUS [10], EtherCAT [11] and Ethernet/IP [12].
The new challenges of distributed control system have been presented by Dripke et alt. [13]. The most important challenge, which have been identify by them are communication among multi-agent systems, time and synchronization due to the parallel computing nature of distributed systems, control tasks, capacity and scalability. In this direction, a lot of new communication protocols for IoT and time synchronization systems, such as OPC UA [14, 15] and TSN [16, 17, 18], are being developed and standardized in order to overcome the existing problems. While in the industrial field there are some examples of the use of the OPC UA protocol for communication [19], in the case of motion synchronization, this protocol is used together with another protocol (EtherCAT) in order to achieve the synchronization of multiple axes [20]. As regards Time-Sensitive Networking (TSN), it is a new communication protocol that is focused on the simultaneous motion synchronization of axes and data exchange [21]. Several companies are developing new devices in order to implement this new protocol [22] and to increase its reliability, security and performances for industrial applications. A milestone of motion control is synchronization among multiple axes [23, 24, 25, 26] that is fundamental in different fields such as Computer Numerical Control (CNC), Surface Mounting Technology (SMT)[27, 28] and automated factories [29]. It is acknowledged that a new step towards the development of distributed control systems is represented by integration and interoperability among industrial programmable controllers of different producers [29]. Some applications already use communication between PLCs or iPCs of different vendors [30] but only for the exchange of information and not for the motion control of motors. Several studies have been carried out to implement distributed controls for motion axes [31, 32, 33]. In all these cases the master control directly commands drivers of the same vendor or drivers directly compatible with the communication network of the master control. This is due to the fact that different vendors use different methods for the control of the dynamics of motors; moreover, motion control is a hard real-time task, which requires a high level of accuracy and reliability. The advantages of a dis-

tributed motion control system, which can be used with drives and motors of different vendors, are the increase of re-usability, reconfigurability and extension of distributed systems. The novel approach of this paper consists in the implementation of a distributed motion control in which a master iPC commands incompatible drivers and motors through a slave PLC. This method differs from the actual ones in which the motion control between different PLCs/iPCs of different vendors is possible only if the two systems use the same protocol for communication and synchronization.

Even in the field of Neural Networks some new finite-time synchronization (FTS) methods have been developed [34]. An example of accurate time synchronization among multiple devices is presented by Shiying et alt [35]. They use the IEEE 1588 protocol for the design of a time synchronization network of experimental advanced superconducting tokamak (EAST) poloidal field (PF) power supply control system. In this case, they synchronize the time among a control system of PFPS on EAST, local controllers, coordination operations of several sets of power supply and DAQ systems (DAS) with the maximum time offset from the master node that never exceeds 50 ns.

This paper is focused on the development of a general master-slave real-time interface between different programmable controllers for the synchronization of motion control by the use of real-time Ethernet communication networks with the IEEE 1588 V2 Time Synchronization protocol [36, 37] and the Ethernet/IP protocol [38].

The hardware architecture, the used communication network and the algorithms, which will be subsequently shown, represent a global solution for motion control among industrial programmable controllers of different vendors.This solution can be used for any type of distributed real-time control systems that include both PLC and iPC devices. The tests have been performed with different PLCs and iPCs, but the vendors' names cannot be shown because of the NDA.

The general solution developed in this paper will be also used with the latest protocols, such as TSN, when they are introduced into the automation field as standard protocols.

Following the method shown in the paper it is possible to overcome the problem of connectivity between PLC and iPC of different vendors without the reduction of the function of one of the two systems. De facto, it is also possible to control functions that require high accuracy and safety such as the motion control of different motors.

The paper is organized as follows. Section 2 shows the notations used in this

paper; Section 3 deals with the existing hardware architectures for the motion synchronization of distributed control systems and the architecture used in this study; Section 4 describes the problem and possible solutions; Section 5 illustrates the simulations performed; Section 6 and Section 7 analyze the solutions adopted for different system architectures; Section 8 explains the experimental results; Section 9 provides a discussion about the work; Section 10 expounds the conclusions drawn at the end of the research.

2. Notations

Cl_M	Clock Master
Cl_S	Clock Slave
Mot_M	Motion Master CPU
Mot_S	Motion Slave CPU
p_i	i-th Set position
v_i	i-th Set velocity
a_i	i-th Set acceleration
j_i	i-th Set jerk
T_{ad}	Actuation delay
T_{stt}	Motion task time of the Mot_S
CUP	Coarse update
D_i	i-th Time difference
T_{mc_i}	i-th Time instant in which Mot_M imposes the set values on its axes
T_{sc_i}	i-th Time instant in which Mot_S imposes the set values on its axes
T_t	Total time
p_{f_i}	i-th extrapolated future position
v_{f_i}	i-th extrapolated future velocity
a_{f_i}	i-th extrapolated future acceleration
T_a	Acceleration time
T_t	Time at constant velocity
p_0	Actual position
v_0	Actual velocity
v_{jog}	The input velocity given to the jog
a_{jog}	The input acceleration given to the jog
e_p	Position error
e_v	Velocity error

3. Hardware architecture

In order to generalize the architecture of the system, the hardware components used are typical of every programmable controller vendor. The only constraints are:

- the Ethernet communication network for the exchange of the kinematic data of motors with a velocity of at least 100Mbps. For some vendors it is directly integrated in the controller card while other vendors need an appropriate card
- an Ethernet card that allows the use of the IEEE 1588 V2 protocol. In some cases it is the Ethernet communication card itself if the system uses a CIP Sync protocol, or in other cases an appropriate card is necessary.

Furthermore, the programmable controller that moves the main motors is called Motion Master CPU (Mot_M), while the one that synchronizes its motors with the master is called Slave Motion CPU (Mot_S). With this method it is possible to synchronize both real and virtual axes. This is very important because in the automation field the main axis of the machine is often virtual and all the other axes, both real and virtual, move synchronously with it.
If the system needs to synchronize real motors, the hardware architecture includes drivers and motors.
If the system synchronizes only virtual axes, drivers and motors are not included in the hardware architecture.
Other studies relating to motor synchronization have been carried out by means of the use of the IEEE 1588 synchronization protocol and Ethernet connections [31]. Even they consider a master-slave architecture, but the main difference lies on the fact that they use only a master controller and some slave drivers of the same vendor as in Fig.1.
On the contrary, this study aims at developing an interface between programmable controllers of different vendors, which control drivers and motors. In this case a master CPU controls different slave CPUs that send the set values to the drive in order to control several motors as represented in Fig.2.

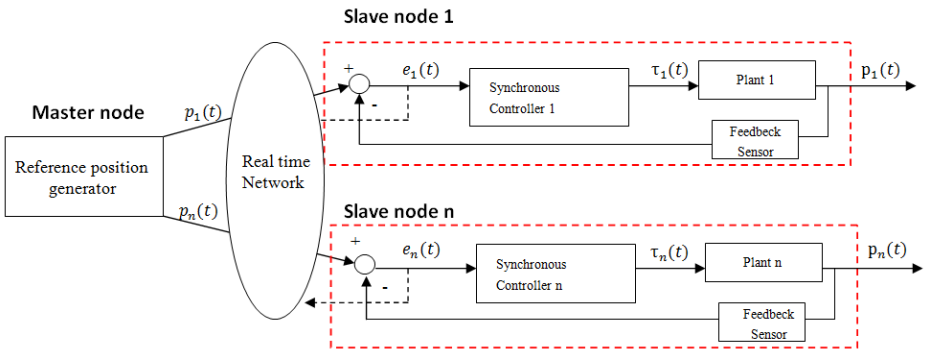

Figure 1: Literature system architecture

4. Problem definition

The synchronization of several axes between programmable controllers of different vendors can be divided into four parts:

1. time synchronization of the system
2. exchange of kinematic data
3. computation of trajectory
4. driver actuation

4.1. Time synchronization

First of all programmable controllers need to have the same time reference so that they can evaluate the kinematic values of the system properly. The clocks of programmable controllers can be very different in accuracy and reliability, consequently it is necessary to use a method which is oriented to find a common clock between the controllers. In order to obtain a high accuracy in motion synchronization, a high accuracy of the global time of CPUs is required.Therefore, the synchronization protocol used is the IEEE 1588 V2 (Precision Time Protocol), because it is one of the most used, it is based on Ethernet communication network and it achieves a clock accuracy $< 100ns$ [36].
This protocol uses a Clock master(Cl_M) that is the source of the synchronization reference and a Clock slave(Cl_S) that is the destination of the synchronization message. In order to synchronize the Cl_S with the Cl_M, the former computes the time difference between its clock and the synchronization message. If the delay is higher than $100ns$, the Cl_S overwrites its time in

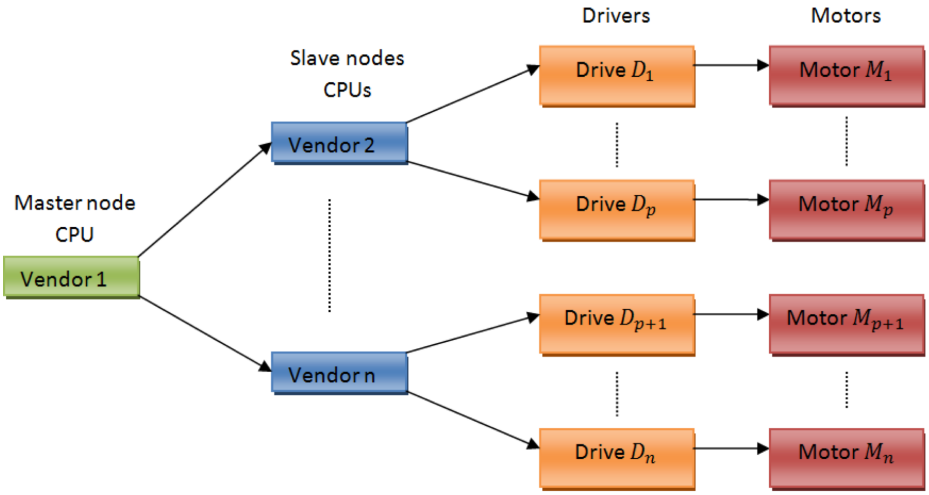

Figure 2: System architecture

order to ensure synchronization, this means that there are some gaps in the time reference of Cl_S. Therefore, it is very important to understand which CPU between Mot_M and Mot_S will be the Cl_M or the Cl_S. The Cl_M chosen between the two different programmable controllers has to be the Mot_S, because it is not possible to control the dynamics of an axis in an accurate way if the reference time is overwritten during the computation of motion.

4.2. Exchange of kinematic data (Communication configuration)

The communication network between the two programmable controllers is based on Ethernet/IP with a Produced/Consumed tag method. This type of communication allows to broadcast and to receive system-shared tags. The choice of this method for the exchange of data is due to the fact that it can be used with typical control networks, it is reliable and it is real-time communication.

The Mot_M can send the synchronization data for multiple axes within a data path. For each axis the shortest data path necessary for the synchronization of the i-th axis (Fig.3) consists of: Time of master commands ($Tmc_i(i)$) and Set Position, Velocity, Acceleration, Jerk ($p_i(i+T_{stt})$, $v_i(i+T_{stt})$, $a_i(i+T_{stt})$, $j_i(i+T_{stt})$) of the axis with a look ahead equal to the Motion Task Time of the Mot_S (T_{stt}).

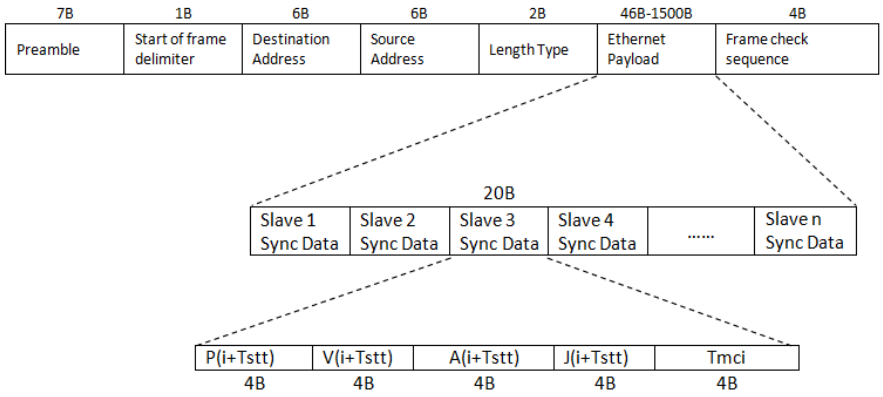

Figure 3: Data path sent to the Mot_S by the Mot_M

4.3. Computation of trajectory

Different vendors use different methods for the motion of axes. Some of them can directly move an axis point by point defining its p_i, v_i and a_i at each motion task. Other vendors, who cannot control a motor point by point, use motion instructions in order to impose the desired trajectory. This case is the most interesting because it requires to impose the desired trajectory by using a predefined motion profile. If the axes to be moved are real, the programmable controller computes their p_i, v_i and a_i and sends them to the drivers. In order to synchronize two or more virtual or real axes between two programmable controllers, the Mot_S needs to take into account the p_i, v_i, a_i and j_i of the Mot_M with respect to the instant of time in which the Mot_M will actuate these commands. Thanks to the time synchronization, the two CPUs have the same clock reference and so it is possible to correlate the time instant in both CPUs. Both in the case of the point-by-point control and in the case of the control obtained by means of motion instructions, the coarse update (CUP) of the motion task and the coarse update of the data sending task of the Mot_M must be shorter than the ones of the Mot_S. This choice reduces the possibility of receiving the same data of motion twice because of the jitter of the tasks that different CPUs have. Moreover, the task in which the motion algorithm is computed must be synchronous with the motion task of the CPU to be sure that the actuation delay (T_{ad}) with which the CPU sends the p_i, v_i and a_i of the system is constant and equal to one CUP of the motion task. To compensate the actuation delay that is equal to one CUP of the motion task of the Mot_S ($T_{ad} = T_{stt}$), the kinematic

values sent by the Mot_M have a look ahead in time equal to T_{stt} and they are $(p_i(i+T_{stt}), v_i(i+T_{stt}), a_i(i+T_{stt}), j_i(i+T_{stt}))$.
The main idea for the synchronization of the two systems is based on the determination of the time difference (D_i) between the time instant in which the Mot_M imposes the set values on its axes (T_{mc_i}) and the time instant in which the Mot_S will impose the set values on its axes (Tsc_i).
With the computation of this delay and thanks to the kinematic data of the Mot_M, it is possible to extrapolate the future set position (p_{f_i}), future set velocity (v_{f_i}) and future set acceleration (a_{f_i}) that the master axes will achieve after the total time $(T_{t_i} = (D_i + T_{ad}))$. The computation of T_{t_i} is afflicted by the jitter of the motion task because the Mot_S can only compute the previous instant of time in which it imposes the commands and not the next one. However, the motion task of the Mot_S is scheduled and so it is almost constant except for the jitter of the task $(T_{t_i} = T_{ad_i} + D_i + jitter_i)$. PLCs and iPCs define a maximum level of the jitter of the task that is different among different vendors and different controllers. The communication jitter does not influence the computation because it is included in the D_i value and so it is determined for each calculation accurately.
In order to synchronize the axes of Mot_S with the axes of Mot_M, the Mot_S has to impose on its axes a trajectory congruent to their actual kinematic values to achieve, at the future time instant, the same p_{f_i}, v_{f_i} and a_{f_i} as the Mot_M reaches.
The extrapolated formulas are the following:

$$p_{f_i} = p_i(i+T_{stt}) + v_i(i+T_{stt})D_i + \frac{1}{2}a_i(i+T_{stt})D_i^2 + \frac{1}{6}j_i(i+T_{stt})D_i^3 \quad (1)$$

$$v_{f_i} = v_i(i+T_{stt}) + a_i(i+T_{stt})D_i + \frac{1}{2}j_i(i+T_{stt})D_i^2 \quad (2)$$

$$a_{f_i} = a_i(i+T_{stt}) + \frac{1}{2}j_i(i+T_{stt})D_i \quad (3)$$

For each motion task the Mot_S has to move its axes satisfying the constraints $p_{f_i}, v_{f_i}, a_{f_i}$ with respect to their actual positions (p_{a_i}), actual velocities (v_{a_i}) and actual accelerations (a_{a_i}).

4.4. Drive actuation

The driver can have different control strategies to perform the best control of motors, but it generally uses only the set positions and the set velocities

given by the controller. The driver interpolates the set values with a frequency of about 8KHz and computes the current necessary to impose the defined motion. For this reason all the synchronization algorithms that have been developed have the aim of generating the correct set positions and set velocities required to synchronize two or more axes.

5. Simulations

Several simulations have been performed in order to demonstrate the correctness of this architecture. The tests consist in the evaluation of the synchronization error between a Mot_M axis and a Mot_S axis with the trajectories shown in Fig.17 and Fig.5. The method used to synchronize the two axes is the aforementioned one. The task time chosen for the motion task of the Mot_M is 2ms and the one of Mot_S is 4ms. The Mot_M sends the data path to the Mot_S every 1ms, while the Mot_S reads the data path every 2ms. A random jitter has been imposed on different tasks as follows:

- $\pm 5\mu s$ Master Send Data Task
- $\pm 40\mu s$ Slave Read Data Task
- $\pm 40\mu s$ Slave Read Data Task
- $\pm 40\mu s$ Slave Actuation Task

The timing model is shown in Fig.4. The simulation results are shown in Fig.5 and Fig.6.

The simulations lead to the following deductions:

- synchronization errors are bounded
- jitter causes noise on the synchronization error
- it is possible to have few points with a high synchronization error of velocity and acceleration in case the motion task of the Mot_M does not sample the punctual variations of the dynamic variables.

6. Point-by-point control

In case the Mot_S uses a point-by-point control it is possible to directly impose p_{f_i}, v_{f_i} and a_{f_i} on the axes so synchronization is insured except for the jitter of the task.

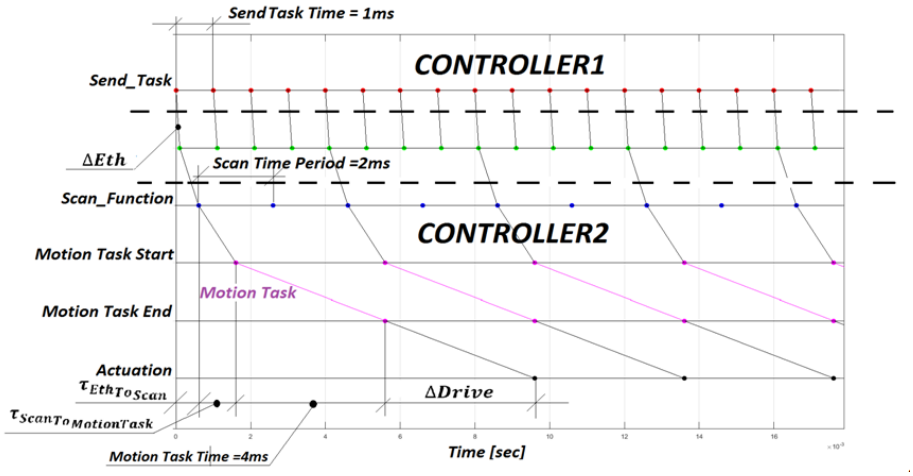

Figure 4: Timing model of the system

7. Motion instruction control

In case the programmable controller can not directly impose the kinematic values on the axes by means of point-to-point control but it can only use motion instructions, it is necessary to use different algorithms to respect the kinematic constraints. In order to develop a global method for the motion synchronization between two different controllers, the motion instruction used to move the axes of the Mot_S is a jog.
The choice of this type of motion instruction depends on the fact that the jog is a trapezoidal velocity profile and it is used in any type of automation PLCs or iPCs. Different vendors give different names to this instruction but, independently from the name, its behavior is the same. The jog instruction allows to carry out a trapezoidal velocity profile (Fig.7) by defining the velocity that the axis has to reach and the acceleration with which it has to obtain the desired velocity.

In this case it is necessary to divide the problem into three different scenarios that depend on the value of the actual velocity(v_0) of the Mot_S axis with respect to the extrapolated velocity(v_f) which the axis has to reach at the next motion task time: $(v_f < v_0)$, $(v_f = v_0)$, $(v_f > v_0)$.

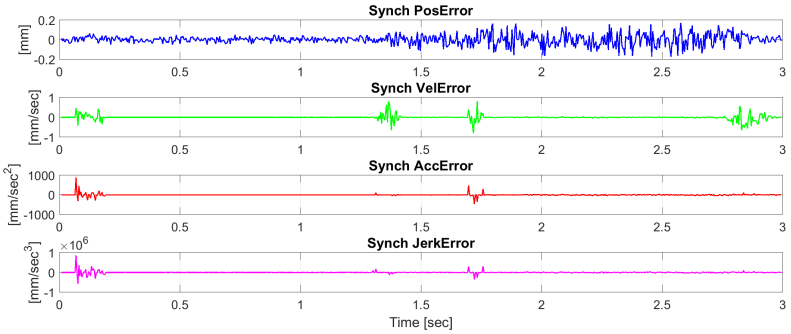

Figure 5: Synchronization error with the first trajectory

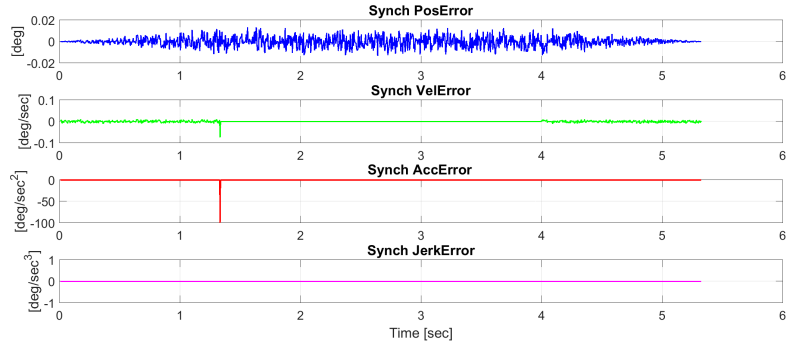

Figure 6: Synchronization error with the second trajectory

In case $v_f > v_0$, the equations describing the problem are:

$$p_f = p_0 + v_0 T_t + \frac{1}{2} a T_a^2 + (v_f - v_0) T_r \tag{4}$$

$$v_f = v_0 + a T_a \tag{5}$$

$$T_t = T_a + T_r \tag{6}$$

where Ta is the time of acceleration, Tr is the time in which the axis has the desired velocity and T_t is the total time. So the solution is the following:

$$T_a = 2 \frac{(p_f - p_0) - v_f T_t}{v_0 - v_f} \tag{7}$$

$$a = \frac{(v_f - v_0)}{T_t} \tag{8}$$

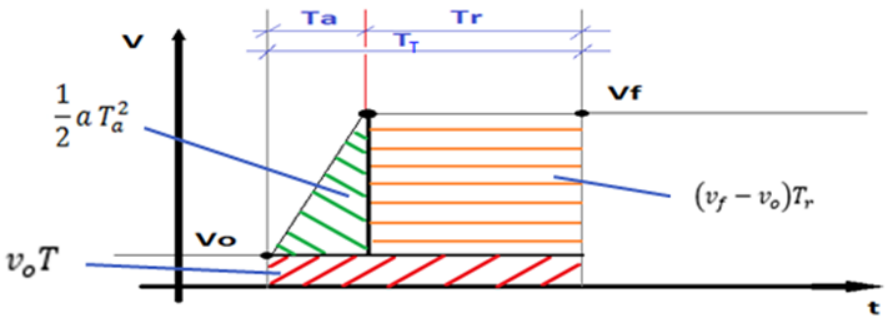

Figure 7: Velocity shape of a jog in which the final velocity v_f of the axis is higher than the actual velocity v_0

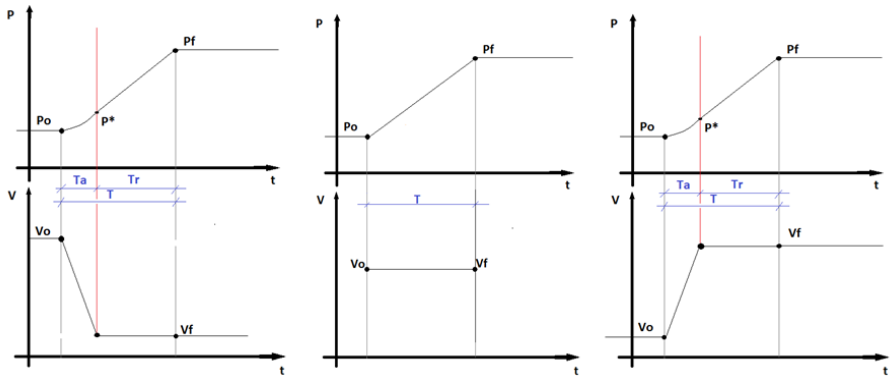

Figure 8: Problem scenarios $(v_f < v_0)$, $(v_f = v_0)$, $(v_f > v_0)$

In case $v_f < v_0$, the solution is similar to the previous one. In the case in which $v_f = v_0$, the unique solution is the following:

$$p_f = p_0 + v_0 T_t \tag{9}$$

$$a = \frac{v_f - v_0}{T_t} \tag{10}$$

The solutions, which also depend on the values of Ta and T, lead to four different cases:

- $0 < T_a < T_T$ the solution always exsists;
- $T_a = 0$ one and only one solution exists and it is not implementable:

$$(p_f - p_0) = v_f T_t \tag{11}$$

$$\frac{\Delta p}{T_t} = v_f \tag{12}$$

- $T_a < 0$ it is impossible to satisfy the position and the velocity constraints at the same time:

$$(p_f - p_0) - v_f T_t < 0 \tag{13}$$

$$\frac{\Delta p}{T_t} < v_f \tag{14}$$

- $T_a > T_t$ it is impossible to satisfy the position and the velocity constraints at the same time:

$$(p_f - p_0) > v_0 T_t \tag{15}$$

$$\frac{\Delta p}{T_t} > v_0 \tag{16}$$

Only in the first case there is always a solution. In the second case there are no implementable solutions because it is necessary to give a new velocity to the axis without any variation of acceleration. In the third case and in the fourth case it is not possible to respect all the constraints so it is necessary to develop an approximated solution. In order to achieve the best synchronization performances with respect to the computational effort, two different algorithms have been developed:

- Discrete approximation
- Linear approximation

7.1. Discrete approximation

With this algorithm, in case $T_a = 0$, the input velocity and acceleration given to the jog are the same as in the previous task. In this way the Mot_M and the Mot_S axes increase their synchronization difference and in the next task T_a will be different from 0. The synchronization error is not high if the time of the motion task is shorter than the dynamics of the Mot_M axes. In the third case ($T_a < 0$), the acceleration time is negative. In the fourth case ($T_a > T$), the acceleration time is higher than the motion task of the system, the inputs of the jog instruction are defined as follows:

$$v_{jog} = \frac{p_f - p_0}{T_t} \tag{17}$$

$$a_{jog} = \frac{v_f - v_0}{T_t} \tag{18}$$

This approximation involves a position error and a velocity error (Fig.9).

To understand if the approximation error is bounded and small enough to allow the application of the algorithm, the position error and the velocity error are computed in the following three cases:

- $v_f < v_0$

$$e_p = \Delta p - \frac{v_f + v_0}{2} T \tag{19}$$

$$e_v = 0 \tag{20}$$

- $v_f = v_0$,

$$e_p = \Delta p - \frac{v_f + v_0}{2} T \tag{21}$$

$$e_v = 0 \tag{22}$$

- $v_f > v_0$

$$e_p = \frac{v_0^2}{2\Delta v} T + \frac{\Delta p^2}{2T\Delta v} - v_0 \frac{\Delta p}{\Delta v} \tag{23}$$

$$e_v = v_f - \frac{\Delta p}{T} \tag{24}$$

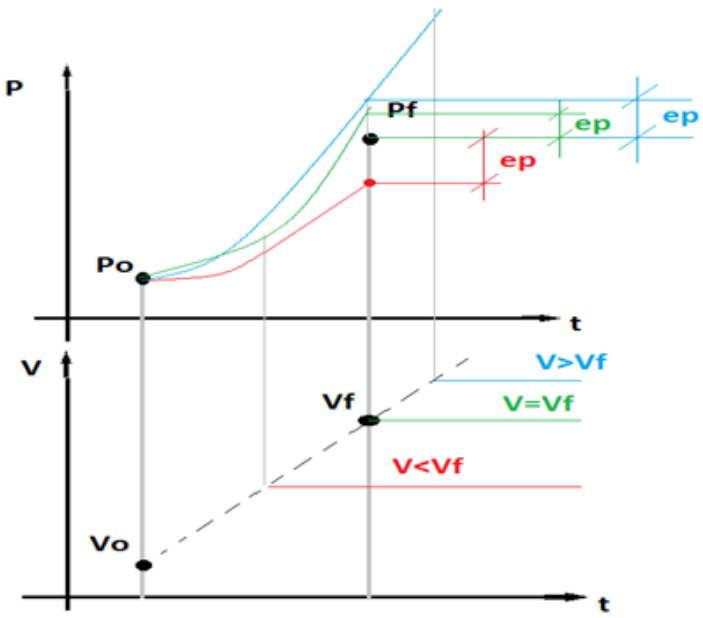

Figure 9: Position error due to the approximation in the cases $(v_f < v_0)$,$(v_f = v_0)$,$(v_f > v_0)$

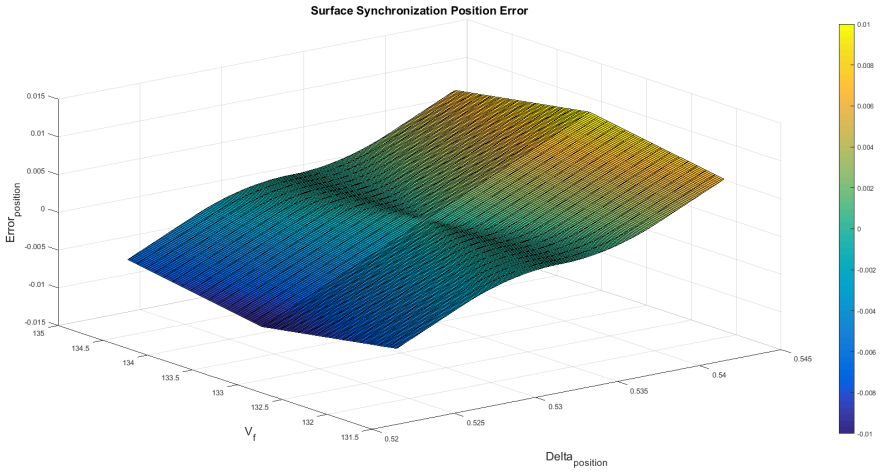

Figure 10: Surface of the synchronization error

Different simulations have been carried out in order to evaluate the position error with respect to Delta position ($p_f - p_0$) and the final velocity v_f. The function is computed in the neighbourhood of the work point with a velocity of about 133,33 deg/sec(Fig.10).

It presents a saddle point in the working point with a position error of about zero, in the other parts the position error is close to zero. The highest position errors can be achieved only if Delta velocity and Delta position are not congruent to each other, consequently only if the kinematic values of the Mot_M axes received by the Mot_S are corrupted.
Another possible problem can be present if the extrapolated velocity is perfectly equal to the actual velocity. In this case the algorithm does not correct any velocity error and the two axes have the same velocity but a very high position error.
In order to solve this problem a very little reduction of velocity is imposed with an acceleration equal to 1% of the actual acceleration of the system. In this way a difference between the extrapolated velocity and the actual velocity of the axes appears in the next motion task. This solution does not produce a high position error if the system has a short motion task time. Fig.11 represents the complete algorithm.

This algorithm has to be computed at each motion task time for each axis to be synchronized. Not all PLCs have a good performance with complex calculations, therefore a simplified algorithm has been developed.

7.2. Linear approximation

The target of this algorithm is to reduce the complexity of computation maintaining good synchronization performances. For each task time this algorithm computes the time difference (D_i), it extrapolates p_f,v_f and a_f and it calculates, by means of eq.17 and eq.18, the input velocity and acceleration to be given to the jog instruction.
The algorithm corrects the position error and the velocity error existing between the actual kinematic values and the extrapolated kinematic values in order to synchronize with the master axis.
However, the algorithm does not distinguish the different kinematic scenarios consequently there is always an approximation error. But, as seen before, the position error is very small if Delta position and Delta velocity are congruent and the computational effort is reduced. This algorithm can be a good trade off between the synchronization performances and the computational efforts if the computational power of the CPU is not too high.

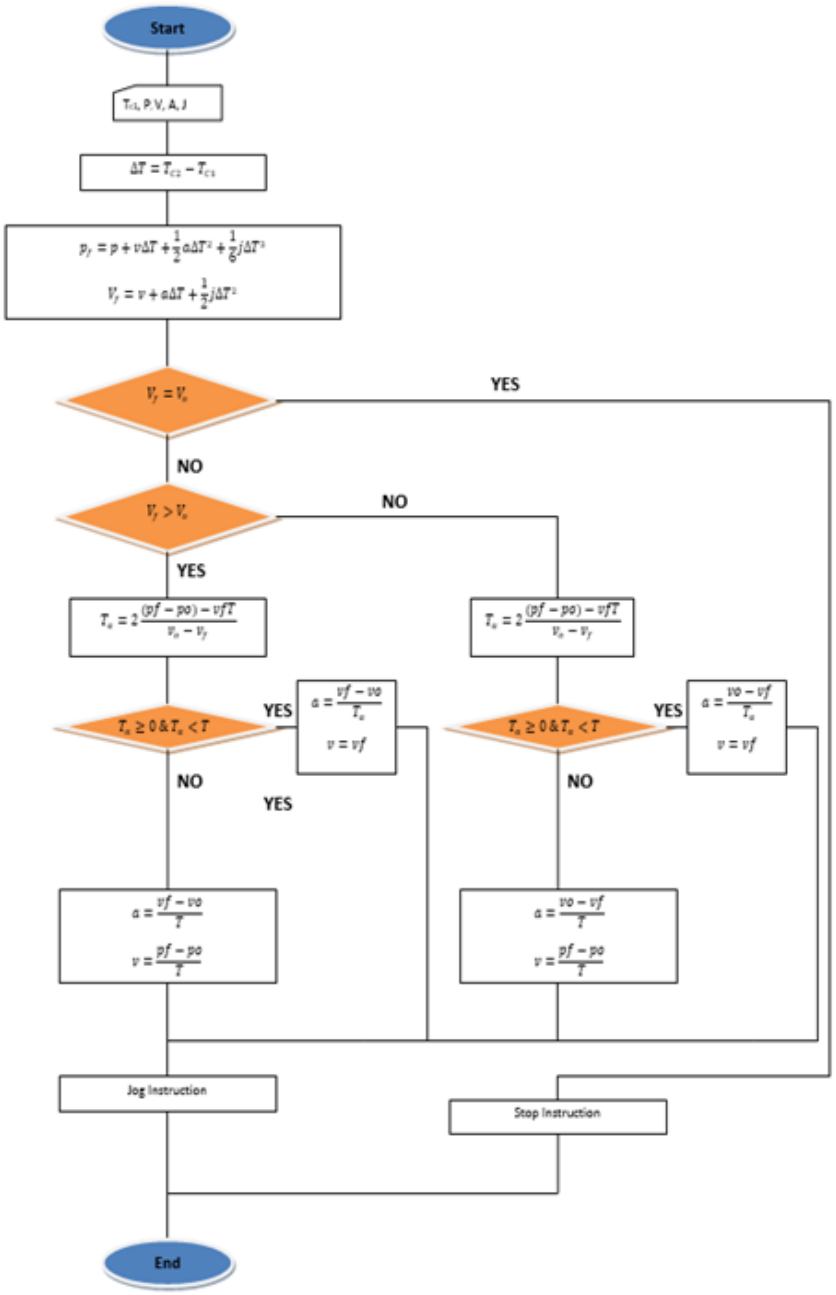

Figure 11: Discrete approximation algorithm flow chart

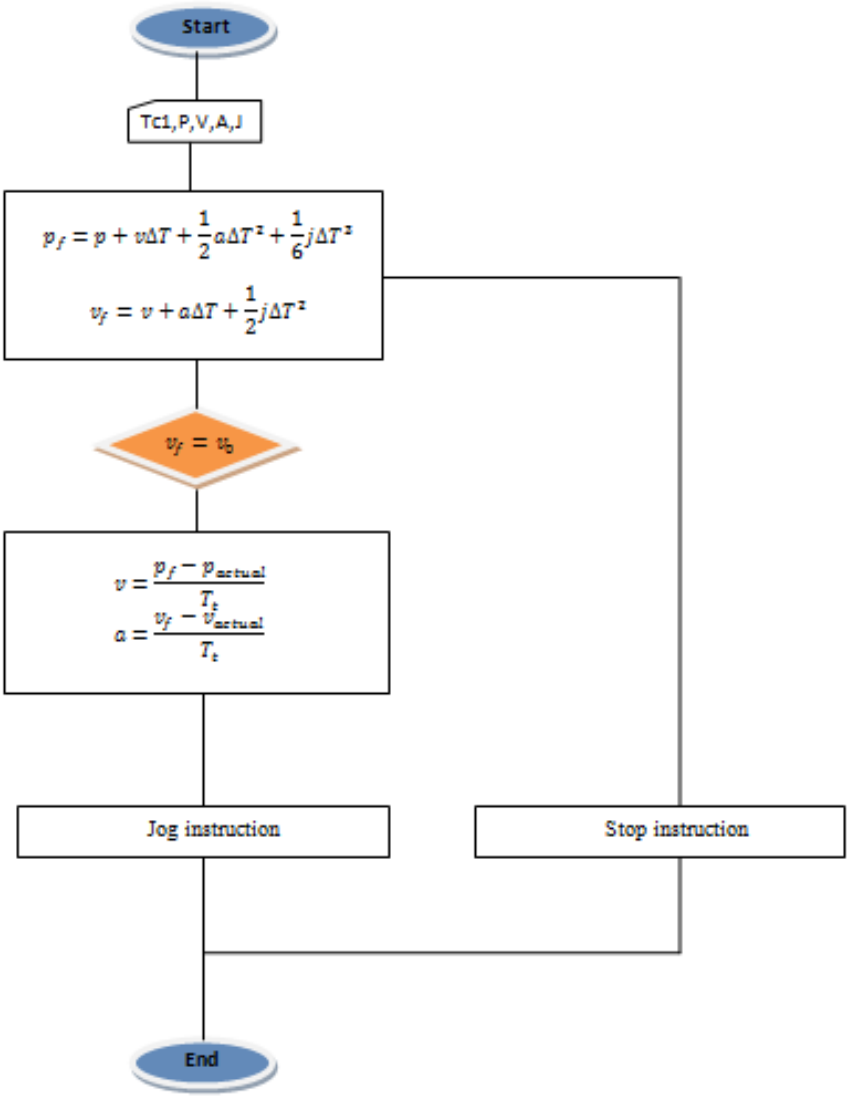

Figure 12: Linear approximation algorithm flow chart

8. Experimental results

The tests have been performed only with PLCs and iPCs that cannot control the kinematics of the axes point-by-point. This choice is due to the fact that in the case of point-by-point control it is possible to synchronize the axes by directly imposing the extrapolated position and velocity of the Mot_M axes. In the case taken into consideration by this study, the time difference between the two systems has to be computed in order to extrapolate p_f,v_f and a_f and to use the Discrete Approximation Algorithm or the Linear Approximation Algorithm required to compute the input velocity (v_{jog}) and the input acceleration (a_{jog}) for the jog instruction and to synchronize the axes.
The tests have been run on virtual and real axes.
The tests on real axes have been conducted in order to understand if the synchronization error can influence the position error or produce vibrations on real motors.
The tests on virtual axes have been performed in order to evaluate the synchronization position error and the velocity position error with respect to different motion profiles. Other tests on virtual axes have been run in order to determine how the variation of the task time of the two CPUs can influence the performances of the algorithms.
It is not possible to show the names of the vendors of PLCs and iPCs used in the tests because of the NDA.

8.1. Tests on real motors

In order to understand if the algorithms can generate noise on real motors, the position error and the velocity error of the distributed hybrid system have been evaluated by comparing them with the position error and the velocity error of the same motor commanded only by one CPU. The motion profile chosen for the test is shown in Fig.13; it consists of a hard dynamics with several parts of accelerating zones and constant velocity zones.

The position error and the velocity error are the same both in the normal case and in the hybrid case.

8.2. Tests on position and velocity synchronization errors

In order to evaluate the synchronization performances of the two algorithms at different velocities and with virtual axes, two motion profiles have been used: the first one is represented in Fig.13, while the second one is

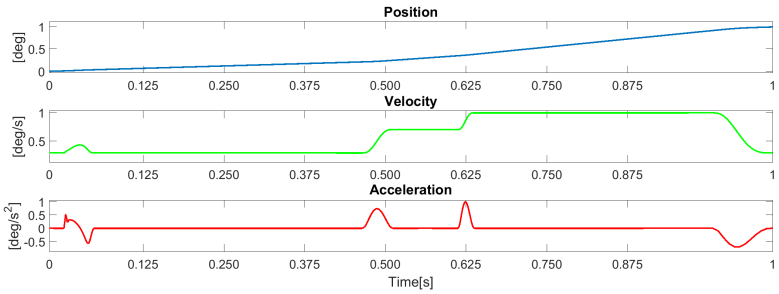

Figure 13: First motion profile

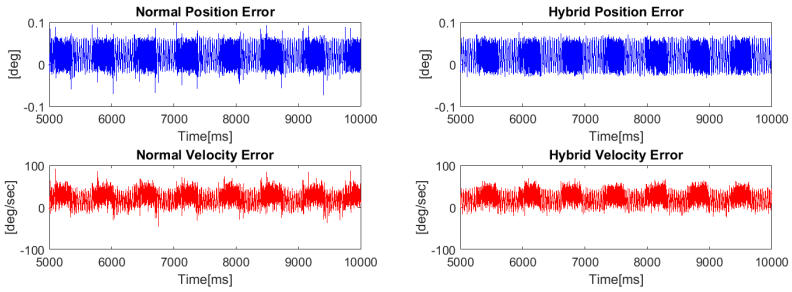

Figure 14: Position error and velocity error in the normal and hybrid configuration

represented in Fig.17.
For the sake of brevity, Fig.15 and Fig.16 show only one test; the variables represent the position synchronization error and the velocity synchronization error. However, the data of all the tests are listed in Tab 1.

- First motion profile

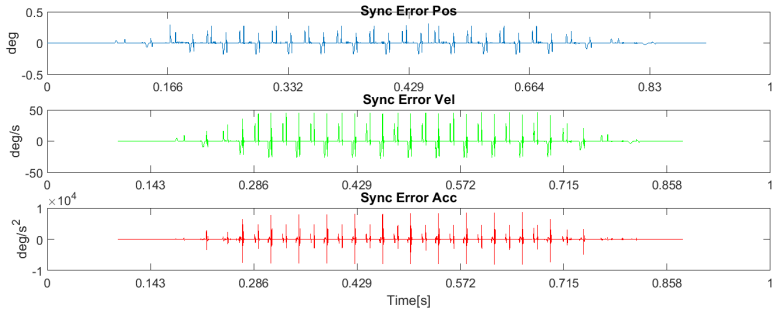

Figure 15: Synchronization error with the Discrete Approximation Algorithm for the first motion profile

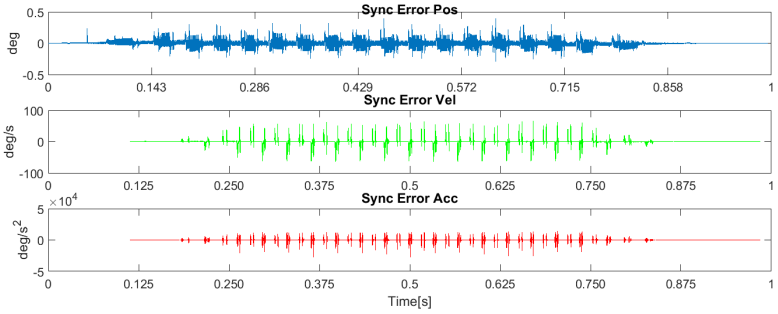

Figure 16: Synchronization error with the Linear Approximation Algorithm for the first motion profile

- Second motion profile

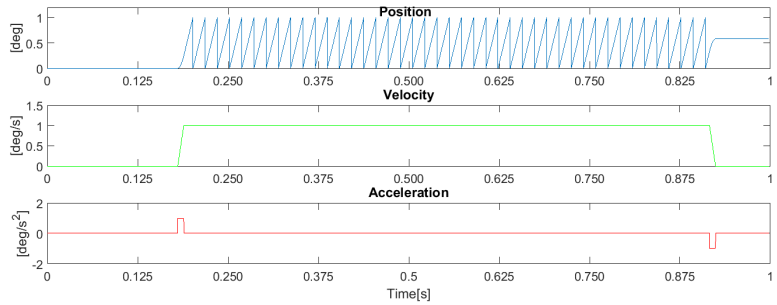

Figure 17: Second motion profile

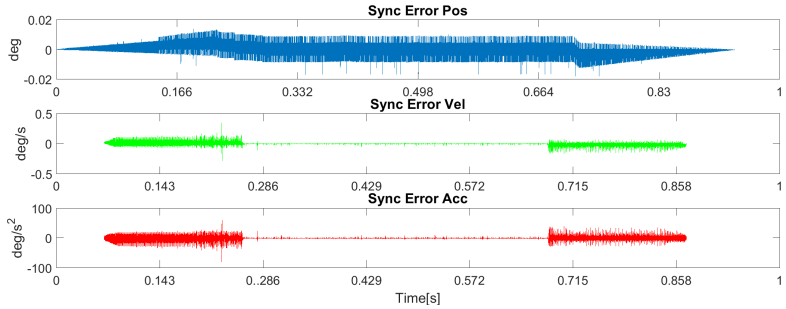

Figure 18: Synchronization error with the Discrete Approximation Algorithm for the second motion profile

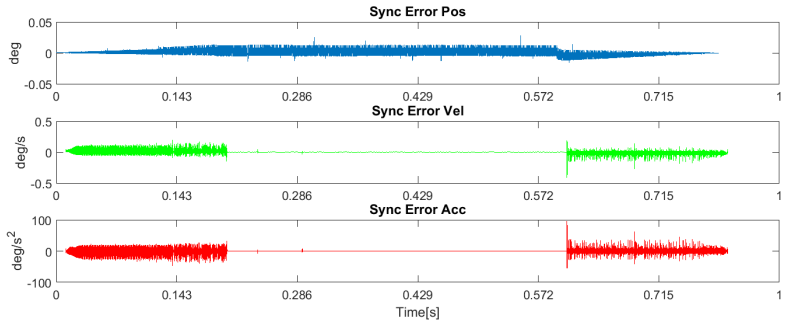

Figure 19: Synchronization error with the Linear Approximation Algorithm for the second motion profile

8.3. Influence of the task time

An important factor for the motion control is the task time of the motion planner of CPUs. The following experiments have been conducted in order to understand if the variation of the task time influences the performances of the algorithms. The tests have been performed for three different task times of the motion planner of the Mot_S:

- 8ms
- 4ms
- 2ms

The task time of the Mot_M is always half of the task time of the Mot_S. The profile used is the same as in Fig.13 at three different velocities:

- 67deg/sec
- 133.33 deg/sec
- 200 deg/sec

The data of all the tests are shown in Tab.2.

9. Discussion

To solve the synchronization problem two possible cases have been taken into account:

- point-by-point control
- axis control by means of motion instructions

In the first case it is possible to synchronize the axes by means of a simple compensation of the time delay between the time in which the Mot_M commands its axes and the time in which the Mot_S performs the same operation. In the second case a trapezoidal velocity profile called jog instruction has been chosen in order to control the Mot_S axes and to synchronize them with the Mot_M axes. This choice has been made because in any PLC or iPC used for the motion control it is possible to define a trapezoidal velocity trajectory by imposing on the axis the velocity and the acceleration to be

reached.
Therefore, two different algorithms have been developed for the solution of the synchronization problem. The main difference between the two algorithms depends on the fact that the Discrete Approximation Algorithm takes into account all the possible kinematic cases of the axes, but it involves a higher computational effort.
On the contrary, even if the Linear Approximation Algorithm has the same behavior in any kinematic case, it requires a lower computational effort.
In order to optimize the algorithms, some tests have been conducted with different task times of the motion planner as the task time is an important factor for the synchronization accuracy. De facto, the shorter the motion task time is, the faster the generation of the command values sent to the axes is. Consequently, the system is more reactive to the correction of the synchronization error and the decrease of the task time reduces the mean and especially the standard deviation of the synchronization error.
These tests have brought to light another parameter that influences the performances. It is the jitter of the motion task of the Mot_S; in order to evaluate its influence, several tests have been conducted on virtual axes following the profile of Fig.13. The data are shown in Fig.20.

Actually a high variation of the jitter involves a higher synchronization error. Even the magnitude of the jitter influences the system: for a high magnitude of the jitter there is an increase of the synchronization error. The problem is that it is not possible to forecast the jitter and to compensate it because it has a randomic behavior. Therefore, in order to reduce its effects, there are two possibilities:

- to choose a CPU with a very low jitter
- to reduce the task time of the motion planner of the Mot_s so that the magnitude of the jitter can be reduced.

10. Conclusions

The first step of this work has involved a wide research on the most used systems of real-time communication and synchronization. For the time synchronization of different CPUs it has been necessary to individualize the hardware and software tools adopted by most vendors of programmable controllers.

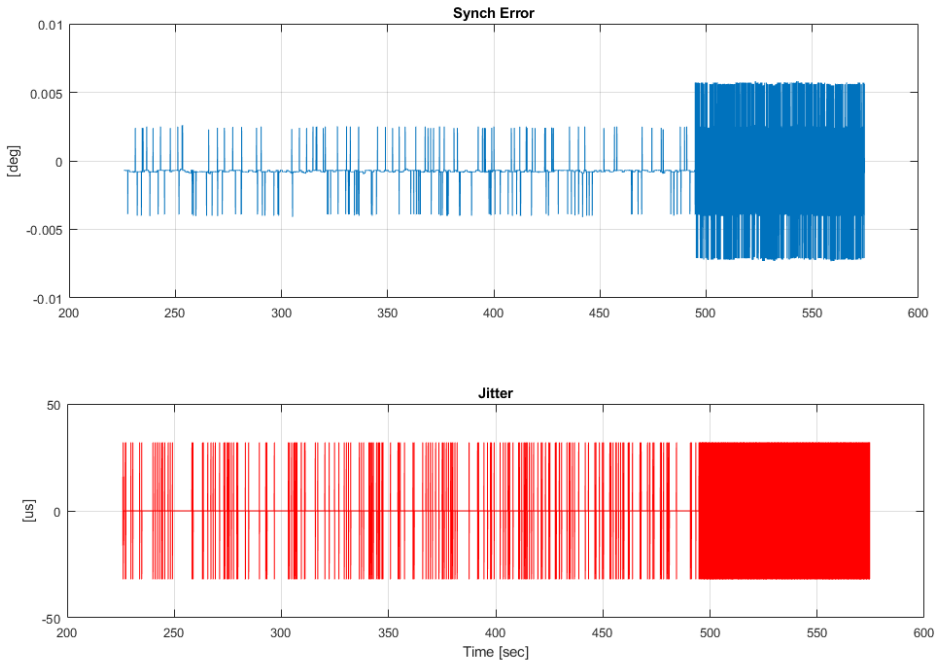

Figure 20: Comparison between the synchronization error and the jitter of the Mot_S motion task

The second step has regarded a research aimed at defining the communication method to be used for the exchange of the kinematic data of the axes between two CPUs.
Two synchronization algorithms have been developed in order to solve the synchronization problem.
Several simulations have been carried out with a defined trajectory of the axes in order to evaluate the synchronization performances of the system.They have shown a position error close to zero along the working point.
Subsequently, the two algorithms have been tested with real and virtual axes. As regards real axes, it emerges that the use of synchronization algorithms do not influence the position error and the velocity error of the real motors. As regards virtual axes, several synchronization tests have been run with different motion profiles and at different velocities in order to evaluate the flexibility of the algorithms. They show that the synchronization error is bounded; it depends on the velocity of the axes and on the motion profile; it is one or two orders of magnitude lower than a common position error of a real motor and it has a tight standard deviation. It means that the algorithms reach good performances and in case of real motors the synchronization error is neglectable.
In the tests conducted with the second trajectory, the synchronization position error of the axes is close to the synchronization position error of the simulations performed with the same profile. Furthermore, a good model of the system has been made.

Compliance with Ethical Standard

Conflict of Interest: The authors declare that they have no conflict of interest.

References

[1] R. Anderl, Industrie 4.0 - advanced engineering of smart products and smart production, in: 19th International Seminar on High Technology.

[2] H. Kopetz, Real-time Systems, Design Principles for Distributed Embedded Application, Kluwer Accademic Pubblication, 2002.

[3] P. Antsaklis, J. Baillieul, Special issue on technology of networked control systems, IEEE, 2007.

Table 1: Synchronization performances

		Discrete approximation				**Linear approximation**			
		P		**V**		**P**		**V**	
		Mean	**Std. Dev.**	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**	**Mean**	**Std. Dev.**
Profile 1	**50 deg/s**	0.001	0.0023	-5E-06	0.016	0.0003	0.0021	9E-06	0.0126
	133 deg/s	8E-05	0.0065	9E-06	0.03	0.0019	0.0073	2E-06	0.03
	200 deg/s	2E-05	0.0083	3E-06	0.0332	0.0023	0.0097	1E-06	0.0324
Profile 2	**50 deg/s**	0.0003	0.0249	2E-04	1.374	0.0019	0.023	5E-05	1.121
	133 deg/s	4E-04	0.0652	0.006	7.4421	5E-04	0.0769	-6E-04	7.6379
	200 deg/s	0.0015	0.0716	4E-04	15.641	-0.003	0.277	0.0053	16.658

Table 2: Influence of the task time in the motion synchronization

		Discrete approx.		Linear approx.	
		Mean	**Std. Dev.**	**Mean**	**Std. Dev.**
2 ms	**67 deg/s**	4.329E-2	4.140E-2	4.885E-2	4.939E-2
	133 deg/s	-2.00E-3	4.068E-2	9.372E-2	1.001E-1
	200 deg/s	-9.905E-4	6.273E-2	7.839E-2	1.255E-1
4 ms	**67 deg/s**	1.289E-2	4.029E-2	1.860E-2	4.353E-2
	133 deg/s	2.437E-2	8.316E-2	3.337E-2	9.599E-2
	200 deg/s	3.397E-2	1.318E-1	4.562E-2	1.644E-1
8 ms	**67 deg/s**	-1.057E-4	3.135E-2	-6.308E-3	5.928E-2
	133 deg/s	-4.168E-2	2.342E-1	-3.671E-2	2.321E-1
	200 deg/s	-1.064E-2	5.615E-1	3.634E-2	6.542E-1

[4] E. R. Alphonsus, M. O. Abdullah, A review on the applications of programmable logic controllers (PLCs), Renewable and Sustainable Energy Reviews 60 (2016) 1185–1205.

[5] J. P. Thomesse, Fieldbus technology in industrial automation, Proceedings of the IEEE 93 (2005) 1073–1101.

[6] S. Vitturi, L. Peretti, L. Seno, M. Zigliotto, C. Zunino, Real-time ethernet networks for motion control, Computer Standards and Interfaces 33 (2011) 465–476.

[7] J. D. Decotignie, Analysis and design of integrated control for multi-axis motion systems, Proceedings of the IEEE 93 (2005) 1102–1117.

[8] IEEE-8023.3, Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, standard by IEEE, 2000.

[9] ODVA, Technology Overview Series: ControlNet, ODVA.org, 2015.

[10] K. Bender, PROFIBUS, the Fieldbus for Industrial Automation, Prentice-Hall, 1993.

[11] M. Knezic, B. Dokic, Z. Ivanovic, Topology aspects in etherCAT networks, in: Power Electronics and Motion Control Conference, pp. T1–1–T1–6.

[12] E. Alessandria, L. Seno, S. Vitturi, Performance analysis of ethernet/IP networks, in: IFAC Proceedings Volumes, volume 40, pp. 391–398.

[13] C. Dripke, A. Verl, Challenges in distributed interpolation with multi-components systems in future-oriented manufacturing units, in: Proceedings of the 47th International Conference on Computers and Industrial Engineering, pp. 215–222.

[14] UA specification, 2008.

[15] OPC UA part 1 - concepts 1.00 specification, 2006.

[16] IEEE std 802.1qbu, 2016.

[17] IEEE std 802.1qbv, 2015.

[18] IEEE std 802.1qca, 2015.

[19] M. Schleipen, OPC UA supporting the automated engineering of production monitoring and control systems, in: 2008 IEEE International Conference on Emerging Technologies and Factory Automation, pp. 640–647.

[20] M. Wang, H. Luo, M. Li, J. Dong, R. Mao, T. Zhao, The application of OPC UA technology in motion control system (2014) 93–95.

[21] M. Gutierrez, R. Dobrin, Synchronization quality of IEEE 802.1AS in large-scale industrial automation networks, in: 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), pp. 273–282.

[22] National Instruments, Designing distributed TSN ethernet-based measurement systems, 2018.

[23] F. F. Perez-Pinal, C. Nunez, R. Alvarez, I. Cervantes, Comparison of multi-motor synchronization techniques, in: Proceedings of the IEEE Industrial Electronics Society, pp. 1670–1675.

[24] Y. Xue, J. Wang, Design of an ethernet/IP-based two-axis servo system, in: IFAC Proceedings Volumes, volume 37, pp. 403–406.

[25] D. Sun, X. Shao, G. Feng, A model-free cross-coupled control for position synchronization of multi-axis motions: Theory and experiments, in: IFAC Proceedings Volumes, volume 38, pp. 1–6.

[26] S. K. Jeong, S. S. You, Precise position synchronous control of multi-axis servo system, Mechatronics 18 (2008) 129–140.

[27] S. S. Yeh, P. L. Hsu, Analysis and design of integrated control for multi-axis motion systems, IEEE Transactions on Control Systems Technology 11 (2003) 375–382.

[28] Y. Xiao, K. Y. Zhu, Optimal synchronization control of high-precision motion systems, IEEE Transactions on Industrial Electronics 53 (2006) 1160–1169.

[29] D. Kolberg, D. Zuhlke, Lean automation enabled by Industry 4.0 technologies, in: IFAC Papers On Line, volume 48, pp. 1870–1875.

[30] H. Li, F. Zhang, J. Zhang, N. Zhang, L. Wang, X. Yang, Research and realization of a spinning machine control and monitoring system by industrial ethernet communication between IPC and OMRON PLC, in: Proceedings of the IEEE International Conference on Information and Automation, pp. 1901–1906.

[31] X. Xu, G. Y. Gu, Z. Xiong, X. Sheng, X. Zhu, Development of a decentralized multi-axis synchronous control approach for real-time networks, ISA Transactions 68 (2017) 116–126.

[32] I. Furstner, L. Gogolak, Synchronizing the motion of multiple electric motors new possibilities for smart motion control, in: IEEE 14th International Symposium on Intelligent Systems and Informatics, pp. 105–110.

[33] C. Pang, J. Yan, S. Jennings, Distributed IEC 61499 material handling control based on time synchronization with IEEE 1588, in: 2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, pp. 126–131.

[34] X. Liu, J. Cao, W. Yu, Q. Song, Nonsmooth finite-time synchronization of switched coupled neural networks, Sci China Inf Sci, 61:5(2018) art.no.052203; IEEE Transactions on Cybernetics 46 (2016) 2360–2371.

[35] S. He, L. Huang, J. Shen, G. Gao, G. Wang, X. Chen, L. Zhu, Time synchronization network for EAST poloidal field power supply control system based on IEEE 1588, IEEE Transactions on Plasma Science (2018) 1–5.

[36] IEEE-1588-2008, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, standard by IEEE, 2008.

[37] J. C. Eidson, Measurement, Control, and Communication Using IEEE 1588, Springer, 2006.

[38] V. Shiffer, The CIP family of fieldbus protocols and its newest memberethernet/IP, in: Proceedings of the Emerging Technologies and Factory Automation Conference, pp. 377–384.