Speed command of a synchronous AC servomotor using a Modicon M340 PLC

Răzvan Drăghici

Petroleum-Gas University, Mechanical and Electrical Engineering Faculty, Automatics and Industrial Informatics Specialty
e-mail: draghicir@gmail.com

Abstract

The paper's purpose is to describe the development of a speed command application for a Modicon M340 PLC. The application was built using the Unity Pro XL dedicated software. This new and modern way of dealing with this kind of problems is very intuitive and easy to understand. Thus, it can be taught in school, thereby preparing the students for the labor market's requirements which are very high in this field.

Key words: *speed command, PLC, Modicon M340.*

1. Introduction

The aim of this paper is to create an application for a Schneider Modicon M340 PLC that will command the speed of a synchronous AC servomotor.

The ma in c haracteristic of s ynchronous AC servomotors is the r elationship between t he frequency of the supply voltage and the speed of the servomotor. Thus, the frequency is directly proportional w ith t he s peed. So, in order to command the sp eed of t he motor, it is needed to control the frequency of the supply voltage. This is realized with a servo drive that is connected to the motor and commanded by the PLC, like in Figure 1.

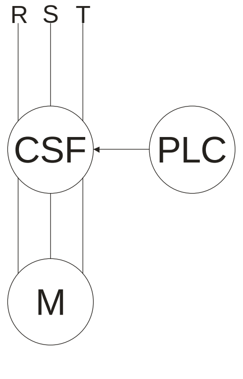


Fig. 1 Classical speed command scheme

2. Experimental details

For the development of the application, a Schneider Modicon M340 PLC was used. This PLC is composed of the following modules: P34 2030 Processor, XBP 0600 Rack, DDM 16025 Relay Input Module, ART 0414 Analog Input Module, EHC 0200 Counter Module.

For this application, besides the processor, only the DDM and EHC modules were used. The command of the motor is realized using a Lexium05 servo drive that communicates with the PLC through a CANopen link.

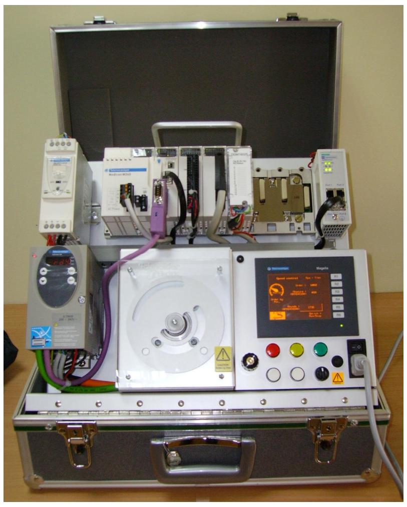


Fig. 2 The experimental stand used for developing the application

Through the application, the user will command the speed of a synchronous AC servo motor. He has the possibility to set a desired value for the speed of the motor using a potentiometer. The set point speed will be shown on a Magelis Human Machine Interface. On the same HMI, some other values are prompted: the actual speed of the motor and the number of rounds the motor has performed. The actual velocity and the number of rounds are measured using an optical sensors system. Also, the motor can be started or stopped using a switch on the stand and the sense of rotation can be modified by pressing a button on the stand. The program was developed using the PLC's dedicated software, Unity Pro XL.

For the development of the application, I followed the next steps: first I had to configure the hardware using the Unity Pro Software, then I had to declare and configure the CANopen Bus, also with Unity Pro, afterwards I had to configure the axes and the servo drive and in the end I programmed the application using the Motion Function Block Library of the Unity Pro software. This provides a simplified access to the basic functions of the servo drive. Thus, motion control variables can be created and axes controlled just by using motion control elementary function blocks.

2.1 Hardware configuration

2.1.1 Procedure for creating a project

When creating a pro ject i t i s im portant to choose the correct h ardware that wil l run the application. So, whe n prompted by the Unity Pro software, the BMX P34 2030 pr ocessor was selected. Forwards, all the other modules, that were p resented earlier, were a dded to t he P LC Bus. In the end, the architecture that was created in the project, matched the architecture of the experimental stand.

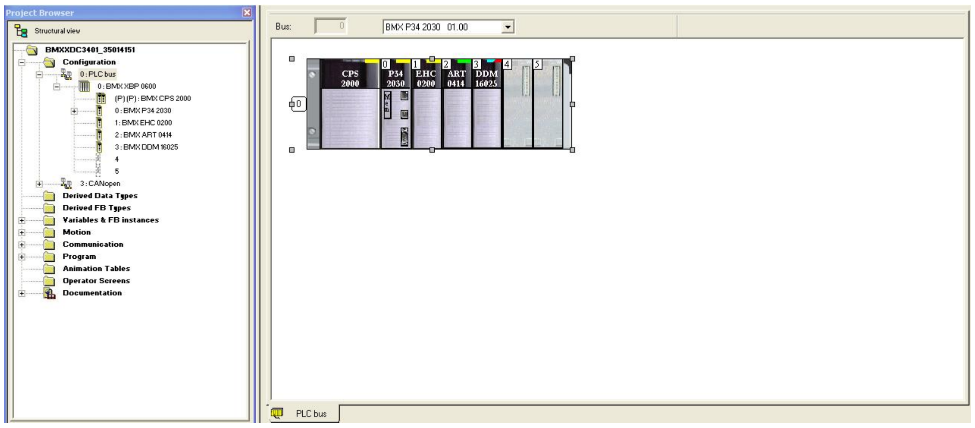


Fig. 3 Setup of the PLC in Unity Pro

2.1.2 Master Task Configuration

In Unity Pro a program can be constructed from:

- A Master task (MAST)
- A Fast task (FAST)
- Sections, that are assigned to one of the defined tasks
- Subroutines sections (SR)

The Master task (MAST) represents the main task of the application program. It i s obligatory and created by default. It is made up of sections and subroutines. Each section of the master task is pr ogrammed in the following lang uages: Ladder D iagram (LD), F unction Block Diagram (FBD), Instruction List (IL), Structured Text (ST) or Sequential Function Chart (SFC).

There are two types of master task execution:

- Cyclic (default selection)
- Periodic (1 to 255 ms)

For this application I selected a cyclic execution.

2.1.3 CANopen Bus Configuration

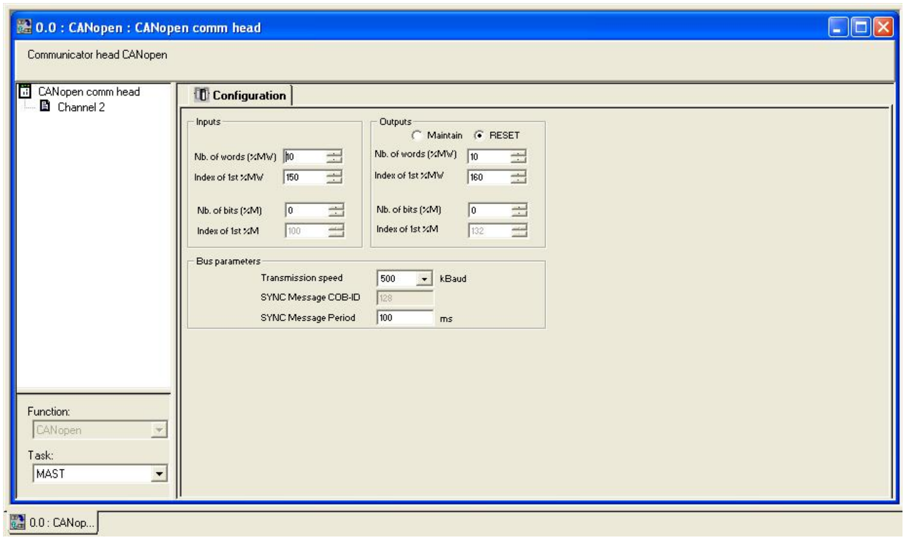
CANopen is a c ommunication protocol and device profile specification for embedded systems used i n automation. Or iginally deve loped for onbo ard automobile systems, the CAN communication bus is now used in many fields, including: transport, mobile devices, industrial control, etc. The CANopen structure consists of a bus master and other slave devices.

The master manages the initialization of the slaves, the communication errors and the statuses of the slaves. Bus ope ration is point to point. At any time, each device can send a request on the bus and the affected devices answer.

2.1.3.1 Configuration of the CANopen Bus Master

In order to configure the Bus master (the PLC), the CANopen port of the CPU must be set up.

In the Unity Pro Project Browser, under the configuration directory, the PLC bus can be found. After accessing the CANopen port, in the Bus parameters area, the transmission speed was set at 500K Baud. In the Task area, the MAST task was selected.



0.0 : CANopen : CANopen comm head

Communicator head CANopen

CANopen comm head
Channel 2

Configuration

Inputs

Nb. of words (%MW)	10
Index of 1st %MW	150
Nb. of bits (%M)	0
Index of 1st %M	100

Outputs

Maintain [✓ RESET]

Nb. of words (%MW)	10
Index of 1st %MW	160
Nb. of bits (%M)	0
Index of 1st %M	132

Bus parameters

Transmission speed	500	kBaud
SYNC Message COB-ID	128	
SYNC Message Period	100	ms

Function: CANopen

Task: MAST

0.0 : CANop...

Fig. 4 CANopen Master configuration screen

2.1.3.2 Configuration of the CANopen Slave

The CANopen slave is the Lexium 05 servo drive. It receives the set point from the PLC through the CANopen connection.

To configure the slave, the CANopen tab in the Unity Pro Project Browser must be accessed. Here, a new slave device must be entered. From the shown list, I se lected the Lexiu05_MFB. The topological address given to the device was 2.

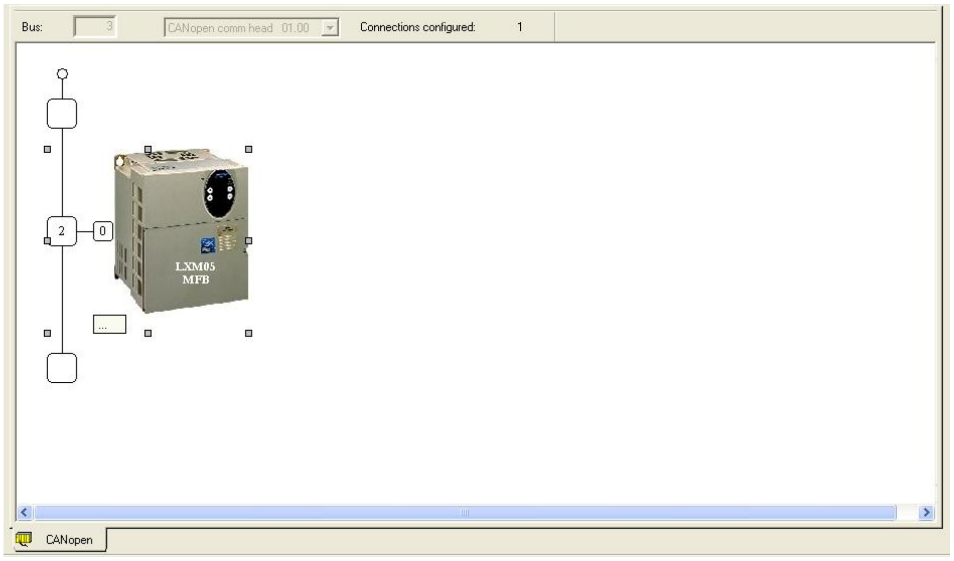


Fig. 5 CANopen slave configuration screen

2.1.3.3 Axis Creation and Configuration

This creation allows to simplify the management and programming of an axis using Unity Pro.

I used the Motion directory in the Project Browser to declare an axis.

Using the configuration panel, the following parameters of the axis were set up:

Name: AXE_MOT

Available drive: Lexium05

Network type: CANopen

Compatible address: \3.2\0.0.0\

Reference of the servo drive: LXM05AD10

Axis reference variable name: AXE_MOT

CANopen handler variable name: Can_Handler_1

For each axis creation, four variables are created:

A Can_Handler-type variable, Can_Handler_1, which can be renamed using the axis directory. It is named after the CANopen manager variable. It must be used in the program as the instance of the CAN HANDLER MFB function block.

An Axis_Ref-type variable, AXE_MOT, which can be renamed using the axis directory. It must be specified in the input parameter for each MFB block used by the axis. It contains all the data needed by the MFBs to work with an axis (logical status, mapping, messaging status, etc.)

A byte table type variable (ARRAY[....] OF B YTE) named by default Recipe_x (where x is a value) but which can be renamed using the R ecipe_x directory. In the application its name is Recipe_1. It is named after the recipe variable which can be seen in the Recipe_x properties of the axis.

An unsigned integer type variable (ARRAY[….] OF UINT) named AxisParamDesc_x (where x is a value) and which may not be renamed. In the application, x=0. This variable is named after the parameter description variable which can be seen in the Recipe_1 properties of the axis.

2.1.3.4 Configuring the Lexium 05

A user interface is integrated in the Lexium 05. With this interface, it is possible to:

- Put the device online
- Configure the device
- Carry out diagnostics

Using the integrated human-machine interface I configured the following parameters :

- CANopen address: \3.2\0.0.0\
- Speed: 500K Baud

2.2 Software configuration

2.2.1 Declaration of Variables

In addition to the variables associated with the axis when it is created in the Motion directory, other variables must be declared.

They must be assigned to:

- Input or output parameters of the MFB blocks
- Input or output parameters of the other modules of the PLC used in this application
- Other parameters that are used for calculations during the application

2.2.2 Programming

Just after declaration and parameter setting of the hardware, motion programming is the second development phase of the application.

Axis programming is divided up into:

- Declaration of variables
- Structured programming in several sections

The table below presents a summary of the program sections that I created:

Section name	**Language**	**Description**
INIT	ST	This section checks if the CANopen link is operational, initializes the communication with the servo drive, and reads the values from the potentiometer and from the optical sensors.
Speed_Control	ST	This section starts the motor, takes the set point from the potentiometer and transmits it to the servo drive as the speed the motor has to reach.

For programming the sections, out of the five programming languages presented earlier, the Structured Text mode was chosen because of the similarity with other programming languages thought in school (like C++).

2.2.2.1 The "INIT" Section

To check if the CANopen communication and that the software and physical configurations are consistent, the CAN_Handler MFB is used. This function must be structured at each UC cycle.

CAN_HANDLER(AXIS:=Axis, NETOP:=NetworkOperational, AXISRD=>AxisReady, ERRID=>ErrorId)

I used the following parameters:

AXIS=AXE_MOT

AXISREADY =CAN_HANDLER_1_RDY

ERRORID = CAN_HANDLER_1_ERRID

Another part of this section is used to acquisition values from the potentiometer and from the optical sensors through the BMX EHC0200 module.

2.2.2.2 The "Speed_Control" Section

In this section, there are three functions that command the servo drive.

The MC_RESET function which is used to initialize the function blocks and to acknowledge servo drive faults.

MC_RESET(AXIS:=Axis, EXEC:=Execute, ERR=>Error, DO=>Done, B=>Busy, ERRID=>ErrorId);

On an Execute rising edge, the parameters are taken into account and the function is executed.

In this case, the function is executed when the PushButton0 is not activated.

The MC_POWER function is used to disable or enable the servo drive.

MC_POWER(AXIS:=Axis, ENB:=Enable, ERR=>Error, ST=>Status, ERRID=>ErrorId);

For the AXIS parameter, the AXE_MOT variable is used and this function is executed when the SwitchButton0 is pressed.

After calling these functions, the speed set point from the potentiometer is read.

Cons_Speed:= Val_Pot ;

The program allows the user to change the sense of rotation of the motor. This can be done by pressing PushButton1.

The next function that is used in the program is MC_MOVE_VELOCITY.

MC_MOVEVELOCITY(AXIS:=Axis, EXEC:=Execute, V:=Velocity, I:=Invert, ERR=>Error, IV=>InVelocity, B=>Busy, AB=>CommandAborted, ERRID=>ErrorId);

The MC_MOVEVELOCITY function is used to execute an endless move command at a given speed. The AXIS parameter is AXE_MOT, the EXEC parameter is attributed to the Speed_Chg variable (that determines a change in the set point speed), the V parameter is the set point speed and I is the sense of rotation.

2.2.2.3 Programming the HMI

I also had to develop a application for the Magelis Human Machine Interface. The program was developed using the Vijeo Designer software. The HMI allows the user to visualize the set point speed and to monitor the actual velocity and the number of rounds the motor has performed.

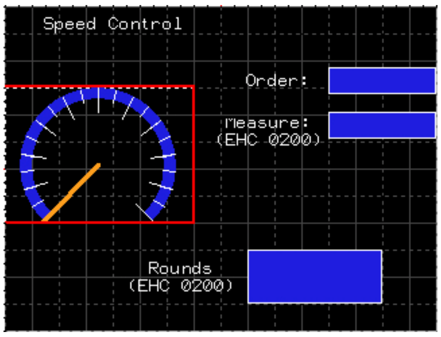


Fig. 6 Screen of the Magelis Human Machine Interface

3. Conclusions

Following the steps that were mentioned in the Introduction, I developed an application for commanding the speed of a synchronous AC motor. With this program, the user can perform the following tasks: turn ON or OFF the motor using the SwitchButton0, set a desired speed for the motor with the potentiometer, change the sense of rotation by pressing the PushButton1 and read the actual speed and the number of rounds the motor has performed with the help of the optical sensors.

With today's PLCs it is simple and quick to build an application with a complex purpose, like commanding a servomotor.

Using the facilities offered by the Unity Pro software, the task of programming the PLC becomes much easier. The MFB function block library is a powerful tool in programming applications that control and command servomotors. This way of programming is intuitive and simple to understand. The weak point in using the MFB library is that the user isn't allow to access the control algorithm.

This application also has an educational purpose. It demonstrates that PLC programming can be taught in school, thus contributing to the professional development of the students.

4. References

1. M ă g u r e a n u , R . – *Maşini electrice speciale pentru sisteme automate*. Editura Tehnică.
2. *Unity Pro XL Help*
3. *Unity Pro Startup Guide*
4. *Unity Pro Programming Guide*

Comanda unui motor sincron folosind un PLC Modicon M340

Rezumat

Scopul acestei lucrări este de a descrie dezvoltarea unei aplicații de comandă a vitezei unui motor sincron folosind un PLC Modicon M340. Programul s-a realizat folosit softwareul dedicat Unity Pro XL. Această metodă modernă de a rezolva astfel de probleme este intuitivă şi uşor de înțeles. Astfel, poate fi predată studenților, pregătindu-i în acest mod pentru cerințele pieței de muncă, piață care are nevoie de specialişti cu cunoştințe în acest domeniu.