Development and practice of universal motion controller communication components for precision measurement systems

Song Xue\textsuperscript{1,a} and Yuewei Bai\textsuperscript{2,b,*}

\textsuperscript{1}Faculty of Intelligent Manufacturing and Control Engineering, Shanghai Polytechnic University, Shanghai, China

\textsuperscript{2}Faculty of Intelligent Manufacturing and Control Engineering, Shanghai Polytechnic University, Shanghai, China

\textsuperscript{a}e-mail: 20191510079@stu.sspu.edu.cn

\textsuperscript{b}e-mail: ywbai@sspu.edu.cn

\textbf{Abstract.} The motion control system and communication system based on the motion controller, the software and hardware communication links, software, and specific hardware are bound together. The coupling between software and hardware is excellent. This paper designs a universal motion controller communication component of the three-coordinate measurement family based on COM component technology and I++ DME protocol, which is used to solve high coupling between measurement software and hardware in the measurement industry. After experimental verification, the COM communication component achieves the expected effect.

1. Introduction

The Motion controller is a critical hardware component of intelligent factory automation equipment, but with the popularization of intelligent chemical factories, control systems based on motion controllers limit the work efficiency of smart chemical factories, so more and more researchers are in research on the design of control systems based on motion controllers, among which communication technology is one of the critical technologies for designing control systems.

In recent years, many scholars have researched this area. Among them, Liangliang Yang et al. have designed a communication module based on an Ethernet motion control card. The upper computer software uses a self-designed communication protocol and a motion control card for data transmission [1]. XU Xiao-ming et al. designed an open CNC system based on PC and motion control card. The host program realizes the control of the hardware by calling the library functions provided by the motion controller [2]. Y. Pititeeraphab et al. designed a mechanical arm control system based on the LEAP motion controller. The upper computer software interacts with it by calling the API function of the motion controller [3].

Through the above research, it is found that most of these researches adopt the Socket communication method and realize the communication between PC upper computer software and motion control card by directly calling the dynamic link library of the motion controller. This leads to a high coupling of software and hardware and product development, maintenance, and maintenance. Integration is more complicated, and so on, aiming at this problem, this article proposes a plug-and-play universal motion controller communication component development method to reduce the coupling between software...
and hardware, aiming at the isolation problem between the three-coordinate measurement software and the measurement hardware system CMM (Coordinate Measurement Machine). The feasibility of this method is proved through engineering practice.

2. Development of supporting technologies for universal communication components

2.1. I++ protocol for CMM software and hardware integration
CMM is an essential piece of equipment used in the precision inspection of intelligent factories. The integration of measurement software and three-coordinate measurement motion controllers was mainly achieved through direct communication in the past, with a high degree of software and hardware coupling. To solve this problem, the CMM industry proposed the I++ DME (Inspection++, Dimensional Measurement Equipment) interface protocol. The I++ DME protocol specifies the communication method between the client and the server. The measurement program acts as a client and communicates with DME-Interface using TCP/IP Socket through a designated port (for example, port 1294). At the same time, the I++ DME protocol also defines a set of communication syntax to ensure the standardization of commands. Table 1 shows a group of typical CMM software and hardware data communication dialogue beginning and ending processes.

| Tag   | Command          | Meaning                                      |
|-------|------------------|----------------------------------------------|
| 00001 | StartSession()   | Client: Start conversation                   |
| 00001 | &                | Server: Received command                     |
| 00001 | %                | Server: Command completed                    |
| 00002 | IsHomed()        | Client: Whether to return to the origin       |
| 00002 | &                | Server: Received command                     |
| 00002 | # IsHomed(1)     | Server: Back to the origin                    |
| 00002 | %                | Server: Command completed                    |
| 00003 | EndSession()     | Client: End conversation                     |
| 00003 | &                | Server: Received command                     |
| 00003 | %                | Server: Command completed                    |

The CMM software supports the I++ protocol and can directly communicate with the CMM motion controller that supports the I++ protocol without repeated development. For CMM motion controllers that do not support the I++ protocol, a universal CMM communication component can be designed to translate the instructions of the non-CMM motion controller into I++ and then send to the CMM software; or to receive the I++ instructions issued by the CMM software, After the translation, it is sent to the CMM motion controller. Therefore, to solve the above-mentioned software coupling problem, multiplexing communication components need to support this I++ industry standard.

2.2. Component Object Model
Dynamic-link library is a commonly used program packaging method, often used to encapsulate shared resources. When multiple programs refer to the same dynamic-link library simultaneously, there is only one copy of the dynamic link library in the memory, which saves memory resources. When CMM software integrates different CMM hardware (here refers to automated CMM equipment), CMM software often needs to call the ordinary dynamic link library provided by different CMM motion controllers. In this case, CMM measurement software often needs to be recompiled. It is unfavorable for the version management of CMM software.

COM (Component Object Model) is an executable program or dynamic link library based on the COM specification. In addition to having the advantages of an ordinary dynamic link library, it also classifies functions by interfaces, which can easily implement inter-process calls and distributed calls.
It is convenient to organize development and module integration. As long as the code of the ordinary
dynamic link library is modified, whether it is to alter the interface or the program logic, the integrated
software needs to be compiled, and the COM component can be plug and play as long as the interface
is unchanged. The logic of modifying the internal code will not affect integration. In addition, COM can
be easily expanded by adding interfaces. Therefore, the development of universal communication
components requires the use of this COM component technology.

2.3. Object-Oriented Programming
The object-oriented development method is a kind of programming thought that conforms to human
thinking. Its core idea is to treat the object as the basic unit of the program and encapsulate the program
and data to improve the reusability, flexibility, and scalability of the software. The purpose of
establishing an object is not to complete a step but to describe the behavior of something in the real
problem-solving action [5].

3. Development of universal communication components

3.1. Design scheme
The communication system based on TCP/IP Socket designed in this paper, combined with I++ DME
protocol, realizes a layer of isolation between CMM software and motion controller and reduces the
coupling of software and hardware. This system comprises CMM software, COM communication
components, motion controllers, CMM, etc. Figure 1 shows the entire composition framework.

According to the above functional requirements, the communication component will be designed as
a dynamic link library conforming to the COM specification based on the I++ DME interface protocol,
using development methods such as object-oriented and software reuse. As shown in Figure 2, the COM
communication component is composed of six COM component objects. The communication process
of the communication component and the interface design of some component objects will be described
in detail below.
3.2. Communication process

As shown in Figure 3, the entire communication process is divided into 15 steps. Next, the whole communication process will be elaborated.

1. The measurement software client sends commands to the COMCommunication component object.
2. After the COMCommunication component object receives the command, it does not process and only passes the received command to the COMCommand component object.
3. The COMCommand component object converts the received command into an I++ command format and then adds it to the command queue defined by the COMDataBase component object.
4. When the I++ command in the command queue is greater than or equal to one, the COMHandlerLogic component object gets the current command.
5. Once the COMHandlerLogic component object obtains the current command, it will feedback "command received" to the COMCommand component object.
6. After the COMCommand component object receives the feedback command, it is converted into an I++ command format (Tag+&) and fed back to the COMCommunication component object.
7. COMCommunication passes the received commands to the measurement software intact.
8. After the command (such as Tag+&) received by the measurement software, the COMHandlerLogic component object will begin to parse the current I++ command and call the corresponding function defined in COMDME.
9. Once the function defined in the COMDME component object is called, the interface function of the COMMotionControlCard component object is bound to be called because all the interface functions of the COMDME component are implemented indirectly by the COMMotionControlCard component object.
10. The COMMotionControlCard component object will generate feedback information during its operation, and the component object will pass its feedback information to the COMDME component object.
11. The COMDME component object feeds its feedback information to the COMHandlerLogic component object.
12. The COMHandlerLogic component object parses the feedback command into three types: error, data, and completion command, and then passes it to the COMCommand component object.
(13) After the COMCommand component object receives the feedback command from COMHandlerLogic, it converts it into the format of the I++ command: error (!), data (#), and completion (%) commands occur, and then the converted command is passed to the COMCommunication component object.

(14) The COMCommunication component object directly sends the feedback I++ command to the CMM software.

(15) Once the completed command (%) is triggered, COMHandlerLogic will delete the completed command in the command queue and finally return to step 4 to continue to obtain the current I++ command.

3.3. Interface Design of COMHandlerLogic Component Object

The COMHandlerLogic component object is the most typical of the six component objects. It obtains and parses the current I++ command in the command queue and then calls the corresponding I++ function. At the same time, the component parses the commands fed back by the COMDME component object into four types, for example, receive commands, data commands, error commands, complete commands, and send them to the client. The function of the interface function defined in Figure 4 will be described in detail below.

![Diagram](image)

Figure 4. Interface Design of COMHandlerLogic Component Object.

Once the function Initialize() is executed, it will start a thread to call the function LogicForProcessingCommand() encapsulated in COMHandlerLogic to process the commands from the command team and the commands fed back to the client. The following will describe the logic of the function LogicForProcessingCommand() in pseudo-code.

1. Function <LogicForProcessingCommand>()(void)
2. Begin
3. If< The number of commands in the command queue >= one > Then
4. {
5.      currentCommand : I++ Command;
6.      currentCommand := The current command of the command queue in COMDataBase;
7.      While(currentCommand != Null) do
8.      {
9.          Send "receive the command" (such as Tag + &) to the client;
10.         Parse and execute the I++ command if the data type (#) is triggered, \n11.         send it directly to the client;
12.         If< trigger I++ error command > Then
13.            {
14.                  Handling error commands;
15.                  Call the function ClearAllErrors() to recover from the error state;
16.                  currentCommand := The current command in the command queue;
17.                  continue;
18.            }
19.         Else
20.            {
21.                Send "Complete Command" (Tag + %) to the client;
22.            }
23.         Delete completed commands in the command queue;
24.         currentCommand := The current command in the command queue;
25.      }
26.      continue;
27.   }
28.   Delete completed commands in the command queue;
29.   currentCommand := The current command in the command queue;
30.}


4. Test
Figure 5(a) shows the entire test environment. The UWC4000i motion controller is used in this experiment because the UWC series motion control card of Lianying Measurement & Control Co., as the first choice for domestic medium and low-end CMM, has certain representativeness. By using the dynamic link library provided by UWC4000i to realize the interface function of the COMMotionControlCard component object, the feasibility of the entire COM communication component is verified.

Figure 5(b) shows the entire experimental process. A simple Winform project is created as a client. The user can manually send commands to the COM communication component server through buttons or dialog boxes and receive commands from the communication component. The content of Figure 5(b) shows that the test results have achieved the expected results.

5. Conclusions
This article uses the UWC4000i motion controller to verify the feasibility of the COM communication component, and the operating results are in line with the expected results, which can isolate the software and hardware. If you use a new motion controller, you only need to add the corresponding interface function to the communication component.

Acknowledgments
This project is funded by the State Key Research and Development Program of China (2019YFB1704300); and received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 734599. The project was partially supported by the Aviation Science Fund of China (No. 20180343002).

References
[1] Yang, L.L., Li, X., Shi, W.M., Lu, W.Q. (2020) Design of communication module of motion control card based on Ethernet. Industrial Instrumentation & Automation., 03: 24-28+59.
[2] Xu, X., Li, Y., Sun, J. (2012) Research and development of open CNC system based on PC and motion controller. Procedia Engineering., 29: 1845-1850.
[3] Pititeeraphab, Y., Choitkunnan, P., Thongpance, N., Kullatham, K., & Pintavirooj, C. (2016) Robot-arm control system using LEAP motion controller. In: International Conference on Biomedical Engineering (BME-HUST). Hanoi. pp. 109-112.
[4] BIEDENBACH, H.M., BRUNNER, J., GLASNER, K., et al. (2020) I++ DME Dimensional Measurement Equipment Interface Version 1.7. https://resources.renishaw.com/details/Specification:+I++DME+specification+version+1.7(156867)(19564).
[5] Lippman, S.B., Lajoie, J., Moo, B.E. (2012) C++ Primer (5th Edition). Addison-Wesley Professional, Boston.