Feature

Interactive motion control using Ch – an embeddable C/C++ interpreter

Stephen S. Nestinger and Harry H. Cheng

The authors

Stephen S. Nestinger and **Harry H. Cheng** are based at the Integration Engineering Laboratory, Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA, USA.

Keywords

Assembly, Motion, Programming languages

Abstract

A flexible agile assembly system requires an open architecture integration environment that is mechatronic device and computer platform independent. An interactive environment allows the users to step through programs and acquire immediate feedback from the system and is most suitable for the development of mechatronic systems used on the shop floor. Ch, an embeddable C/C++ interpreter, was developed for mechatronic-independent task-level programming. An experimental mechatronic system with an IBM 7575 Robotic Arm and a National Instruments' motion control board has been developed to demonstrate the capabilities and the ease in integrating mechatronic devices in Ch, which is freely available for downloading.

Assembly Automation
Volume 24 · Number 2 · 2004 · pp. 152–158

· ISSN 0144-5154
DOI 10.1108/01445150410529928

1. Introduction

In the world of manufacturing, the assembly of new products is not as easy as it used to be. Typical assemblies nowadays have hundreds or more parts and so require more operations. Automating the operations can become challenging when dealing with different mechatronic devices from different manufacturers since each manufacture often has their own interface language. Industry solution providers often find themselves in a dilemma. They can either purchase all mechatronic equipment and devices from one manufacturer, limiting the overall design possibilities or purchase the equipment from multiple manufacturers, relying on the production engineers to work out how to operate each device. One way of overcoming this problem is utilizing a mechatronic independent motion language. A congenial programming language environment is critical for rapid integration of mechatronic devices into an assembly system.

Integrated mechatronic systems are typically based on an open-architecture system. By definition, an open-architecture provides open access to real-time data and information that can be used to allow vertical integration of machines within the enterprise, and is critical to agile manufacturing. The use of open-architectures is seen as vital to improve manufacturing within numerous industry groups (Cheng, 2001; Wright and Greenfeld, 1990).

Research and application experience of assembly systems indicate that an ideal programming language for assembly systems must be a sophisticated computer programming language. The language should appeal to both sophisticated expert programmers and novice users. In typical cases, sophisticated users will write high-level functions that can be readily used by less experienced users. The language should support multi-task processing allowing for the control of multiple mechatronic systems and should provide an open-architecture environment. The language should also include a user-friendly graphical interface. Ch is a C/C++ interpreter that was originally developed by Cheng (1993, 1997) based on the need for a mechatronic-independent task-level

Assembly Automation
Volume 24 · Number 2 · 2004 · 152–158

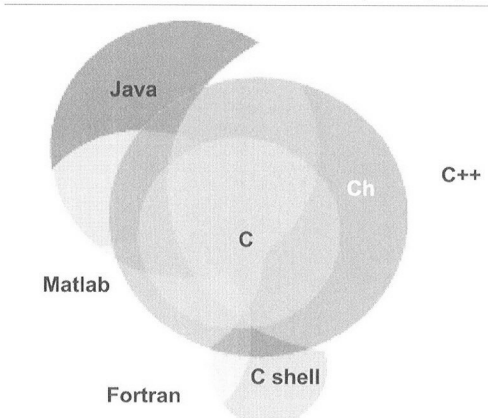
programming environment. Ch is an open architecture integration language environment for the integration of mechatronic systems in agile manufacturing, interactive motion control, rapid prototyping, web-based remote motion control, and as a learning tool for motion control. Ch supports GTK+ and OpenGL with a user-friendly graphical interface across different platforms. Note that an entirely visual based programming environment without a base of procedural programming language is difficult to program for complicated manufacturing tasks, especially for sensor fusion. Ch is available free for downloading from SoftIntegration (SoftIntegration, Inc., 2004a).

This paper discusses the interactive open architecture environment, Ch, and its multitude of applications. It then discusses the ability to integrate new mechatronic device through the use of Ch SDK. An example using an IBM 7575 Robotic Arm Manipulator and one of National Instruments widely used motion control boards is given.

2. Ch – a C/C++ interpreter

Ch is an extension and enhancement of the most popular Unix/Windows/C computing environment. The relation between Ch and other major computer programming languages and software packages can be shown in Figure 1. Ch is a superset of C. Many features first implemented in Ch were added to the latest C standard called C99. These features include

Figure 1 Relation of Ch with other major languages



complex numbers, variable-length array (VLA), binary constants, and function name ___func___. Ch also supports computational arrays as first-class objects as in FORTRAN 90 and MATLAB for linear algebra and matrix computations which greatly simplifies the forward and reverse kinematics required for multi-linkage motion system. Furthermore, Ch supports all features of the ISO C90 standard ratified in 1990.

From a user's application point of view, Ch is computer platform independent, mechatronic device independent, and mechatronic system independent. A program developed in one platform can run on any other platform. This allows Ch users the ability to port programs to any computing environment instead of compiling and linking computer programs on different systems. Experienced C programmers can readily use the system without retraining. Ch also has networking features such as Berkeley sockets and restricted/safe Ch shell that provide both flexibility and security for the remote operation of mechatronic systems. Ch is especially suitable for Web-based client/server computing. Ch programs can be used through a common gateway interface (CGI) in the Web server and as dynamic applets executed through the Web browser. Some salient and important features of Ch for mechatronic integration for assembly systems are as follows:

- Deterministic for real-time computing;
- Interpretive;
- Interactive user interface;
- Superset of an established language;
- Based-on an open language with international standard;
- Object-based;
- Graphic user interface;
- Supported in heterogeneous platforms;
- Code portable across-different platforms;
- Secure network computing;
- Advanced numerical computing with matrix; and
- Interface binary objects.

Using Ch's plug-and-play integration language environment, a new mechatronic system can be quickly developed in the same manner as a UNIX or Windows software installed in a computer. As a superset of C, Ch retains

C's low-level features for interfacing to hardware. Because Ch is implemented as a shell, a large collection of existing utilities and C programs can be readily used for integration of mechatronic systems. Based upon the concept of shell programming, the Ch integration environment is open, modular, and scalable. Software for integration of mechatronic systems are not created by writing large programs starting from scratch, but are created by combining relatively small components. Functions, commands, and scripts are the basic building blocks for the integration of mechatronic systems. These programs can be created and executed dynamically based on the external sensory information, which can be critical to the intelligence of a mechatronic and assembly system.

3. Interactive motion control in Ch

Mechatronic systems have become a valuable part of every manufacturing industry. Each mechatronic system utilizes its own language or communication device in order to be controlled. For example, typical robotic systems come prepackaged with their own robotic language and many motion controller boards available on the market today utilize their own language to control any system attached to it. Creating a workcell made of multiple mechatronic systems can be difficult and tedious due to all of the different languages that must be utilized. Manufacturing companies typically purchase equipment from the same company which reduces the need to learn new mechatronic system languages. This can impede the abilities of the manufacturing company by limiting their ability to purchase mechatronic systems that can further help their business or to improve it. Using Ch, all mechatronic systems can be controlled under the C/C++ language and one environment.

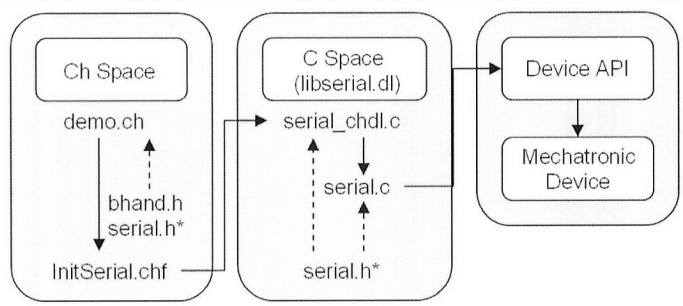
The control of any mechatronic system is done using the available C libraries and functions from the system's manufacturer. Ch Software Development Kit (SDK) (SoftIntegration, Inc., 2004b) allows users to port the C libraries and functions into Ch space by creating Ch packages that bind to the C space libraries and functions. Once that is done, the functions can be called under Ch space with the same ease as user defined functions. Figure 2, shows the architecture overview of how Ch communicates with a mechatronic system.

Once the libraries and functions have been ported over to Ch and the Ch packages created, there are three different ways users can communicate with their mechatronic devices. The first way is to utilize Ch's ability to interpret C/C++ at the prompt. Users can simply type in function names and commands at the prompt and control the mechatronic system. This gives users interactive control of their mechatronic system making debugging and problem-solving easier. The second way, after testing a system by interactively controlling it, is to utilize Ch's ability to interpret C/C++ by placing the user's interactive code into a C/C++ file and running that file as a program. C/C++ files can be readily run at the prompt by typing in the name of the file. The third way is to then compile their programs into a binary executables allowing users to create optimized programs. An example using Ch for motion control is given. The example deals with a National Instruments' Motion Controller Board and an IBM 7575 Robot Manipulator.

4. Motion control using NI-Motion

NI-Motion is a motion control line developed by National Instruments that offers a complete selection of motion control software, controllers, and power drives. NI-Motion controllers range from high-performance controllers with full-feature

Figure 2 The architecture of how Ch communicates with a mechatronic device



capabilities to lower-cost controllers used for point-to-point motion applications. These boards are also available for a multitude of platforms. NI-Motion motion control software comes with LABVIEW, Motion Assistant, a motion control module for Measurement Studio, and a wide array of drivers and application development tools for Windows 2000/NT/Me/9x. FlexMotion is the driver and application development software bundle used in conjunction with the FlexMotion controllers. NI-Motion also offers a universal motion interface (UMI) board to use in conjunction with drives not manufactured by National Instruments.

The Ch NIMotion package is an open source Ch binding to National Instruments' NI-Motion FlexMotion C library (National Instruments, 1999). With Ch NIMotion, all C (or C++) programs utilizing functions in NI-Motion FlexMotion C library can readily run in Ch interpretively without compilation. The Ch NIMotion package is available at the Ch NIMotion Web site (http://ChNIMotion.sourceforge.net).

Figure 3 shows a general overview of the NI-Motion/IBM 7575 Robotic Arm Manipulator architecture.

An example of interactive motion control using the Ch NIMotion package in the Ch command shell is given below. These commands will instruct the first axis to move to position 10000. The initialization program supplied with the package needs to be run first. The initialization program will initialize the board and enable the axis used.

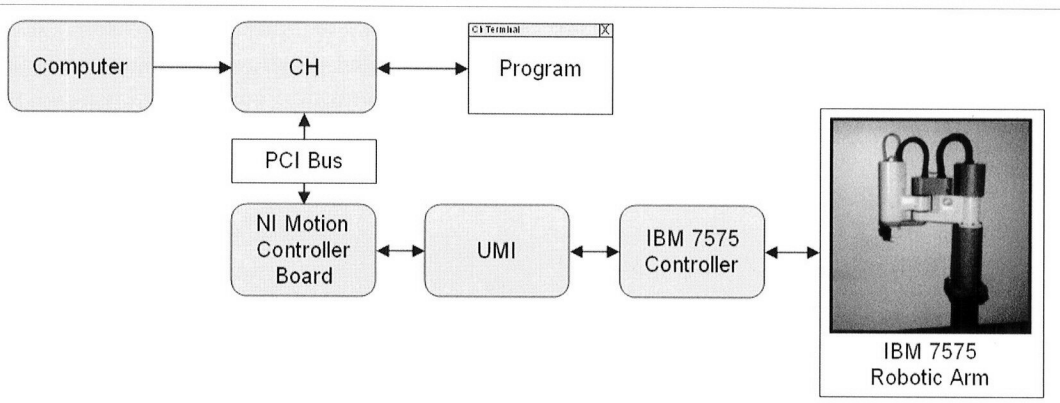
At the root directory type (hit enter at the end of each line):

```
> parse c:/ch/package/nimotion/include/
  flexmotn.h
> u8 boardID = 1
> u8 axis = 1
> i32 targetPosition = 10000
> i32 CurrentPosition
> flex_read_pos_rtn(boardID, axis,
  &CurrentPosition)
> CurrentPosition
> flex_load_target_pos(boardID, axis,
  targetPosition, 0xFF)
> flex_start(boardID, axis, 0)
```

These few lines of code show the ease of use in controlling a mechatronic system under Ch. If users prefer, they can create program files to run later or compile for faster runtime capabilities. The example program shown below will move axis #1 of the IBM 7575 robotic arm manipulator to a position specified by the user at the prompt.

```
// Preprocessing
#include "flexmotn.h"
#include <stdio.h>
#include <chplot.h>
#define MAX 1000
int main(void)
{
  // Variable Declaration
u8 boardID = 1;
u8 axis = 1;
i32 status;
i32 targetPosition = 0;
u16 axisStatus;
i32 CurrentPosition;
```

Figure 3 The NI-Motion/IBM 7575 Robotic Arm Manipulator setup architecture



Assembly Automation
Volume 24 · Number 2 · 2004 · 152–158

```
u16 numberOfSamples = MAX;
u16 timePeriod = 5;
i32 velocity[MAX];
i32 position[MAX];
i32 XAxis[MAX];
i32 returnData[2];
u16 axisMap = 0x0002;
int i;
int j;
status = flex_read_pos_rtn(boardID, axis,
  &CurrentPosition);
printf("Current position of Axis 1: %d",
  CurrentPosition);
printf("\n\nEnter Target Position: ");
scanf("%d", &targetPosition);
fflush(stdin);
printf("\n");
status = flex_load_target_pos(boardID, axis,
  targetPosition, 0xFF);
status = flex_acquire_trajectory_
  data(boardID, axisMap,
       numberOfSamples, timePeriod);
status = flex_start(boardID, axis, 0);
do{
     status = flex_read_pos_rtn(boardID,
     axis, &position);
     printf("\rAxis %d position: %10d", axis,
     position);
     status = flex_read_axis_status_
     rtn(boardID, axis, &axisStatus);
} while (! (axisStatus &
(NIMC_MOVE_COMPLETE_BIT \
          NIMC_AXIS_OFF_BIT)));
i = 0;
while((status = =NIMC_noError) &&
(i < MAX)){
status = flex_read_trajectory_data_
rtn(boardID, returnData);
  position[i] = returnData[0];
  velocity[i] = returnData[1];
  XAxis[i] = i;
  i++;
}
i32 X[i];
i32 Vel[i];
i32 Pos[i];
for(j = 0; j < i; j++){
  Pos[j] = position[j];
  Vel[j] = velocity[j];
  X[j] = XAxis[j];
}
plotxy(X[ ], Pos[ ], "Position vs. Time",
  "Time", "Position");
plotxy(X[ ], Vel[ ], "Velocity vs. Time",
  "Time", "Velocity");
printf("\n\nFinished.\n");
exit(0);
}
/* End of Program */
```

The preprocessing section of this program states which header files to include. The *flexmotn.h* header file contains all of the function prototypes available from the FlexMotion library. The *chplot.h* header file contains the plotting class available in Ch. The plotting class is used to plot the output of the program on the screen or save the plot to a file. The *MAX* variable is the amount of data samples to take and store during the motion.

The variable initialization section is used to setup the system required information and create any required variables. The control board identification number, *boardID*, is used to tell the system what board number to send the commands to. In this program, the *boardID* is assumed to be 1. The axis variable, *axis*, is used to tell the system what axis is to be programmed. This program assumes only the first axis will be used. The status of the system is returned after each function call and is stored in the *status* variable. The *status* variable is constantly checked for signs of an error. The *targetPosition* variable is used to store the target position specified at the prompt. The *axisStatus* variable is used to store the status of the axes for error control. The *CurrentPosition* variable is used to store the current position of the axis. The *numberOfSamples* variable is used to tell the motion control board how many position and velocity samples to take during the motion. The *timePeriod* variable is used to tell the motion control board at what interval to take data at the sample points. The *velocity[MAX]*, *position[MAX]*, and *XAxis[MAX]* arrays are used to store the samples recorded during the motion after they have been retrieved. The *returnData[2]* array variable is used to retrieve the sample data. The *axisMap* variable is a bitmap of the axes used. The integers *i* and *j* are variables used for looping purposes.

Assembly Automation
Volume 24 · Number 2 · 2004 · 152–158

Each function will return the status of the board after the function is called. If an error had occurred, the error number would be stored in the *status* variable for lookup purposes. This program assumes that no errors will occur. The variable, *status*, is only used when attaining the data points acquired during the motion of the system.

The program starts by reading the current position of the system through the function *flex_read_pos_rtn*. The function takes in the board ID, the axis to be controlled, and the address of the variable to store the current position. The program then asks the user to input a target position. After acquiring the target position, the function *flex_load_target_pos* loads it into the specified board for the specified axis. The *flex_acquire_trajectory_data* function enables the data acquisition ability of the board allowing it to store the velocity and position of the system in motion. The function takes it the board ID, axis map, the number of samples to take, and the time period of one sample. The *flex_start* function starts the motion on the axis specified through the axis map. A do loop is initiated to continuously read the position of the axis and print it to the screen until the motion of that axis is completed. The function *flex_read_pos_rtn* reads the current position of the system and stores it in the current position variable. The function *flex_read_axis_status_rtn* reads the current status of the axis. When the motion is completed, the do-while loop is terminated. A while loop is then initiated to acquire all of the stored data samples. The while loop will discontinue when the status of the board gives an error indicating that there are no more data points left or the maximum allowed array subscript value defined in the preprocessing section. The variable *returnData* sent with the function *flex_read_trajectory_data_rtn*, which acquires the data points from the controller board, stores the velocity and the position of the system at each time interval. The velocity and position is then broken up into two different arrays and an *x*-axis array is created for plotting use. A for_loop is then initiated that will remove any continuous zeros at the end of the data points collected. Continuous zeros are created when the time interval and time span specified last longer than the motion. The continuous zeros are removed to show a nicer plot of the velocity and position. The *plotxy* function calls Ch's internal plotting class to plot the results.

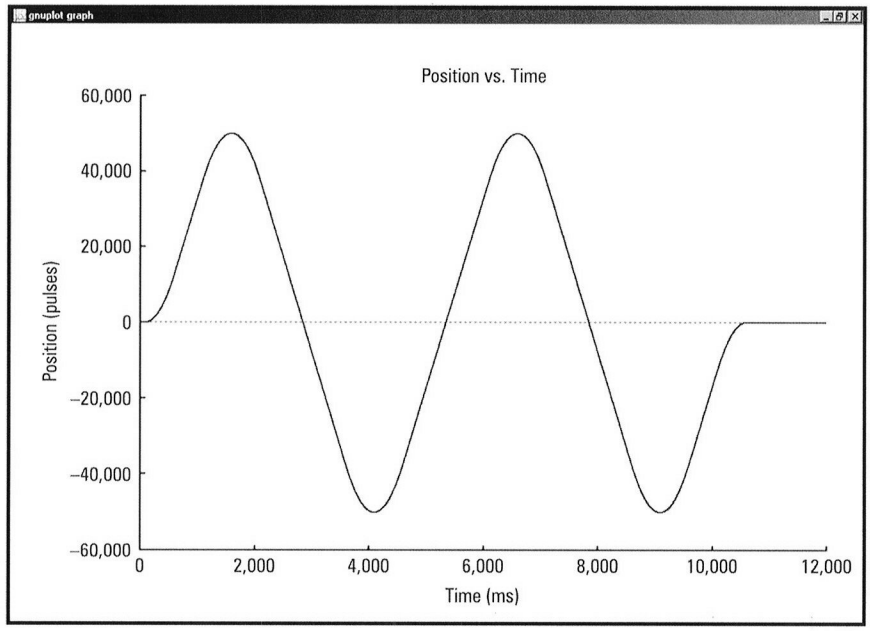
Utilizing Ch's plotting abilities and NI-Motion's ability to store position and velocity information shown in Figure 4, were created. The plots show the motion of a system over time, the first giving the position of the system and the second giving the velocity.

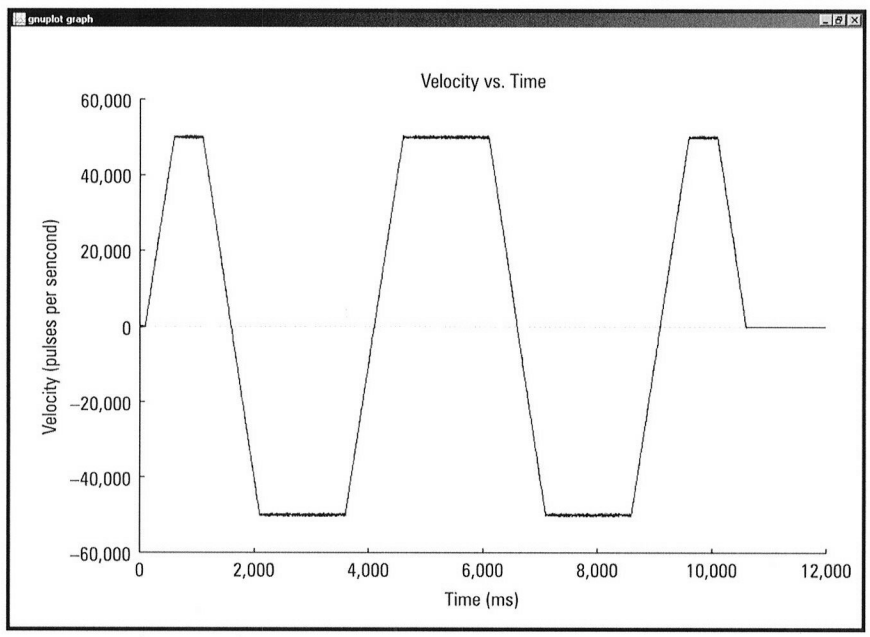
Utilizing the remote capabilities of Ch, a Web-site can be created to give any user using a standard browser the ability to control a motion system remotely while still attaining any desired feedback such as the previous plots. Web-based remote motion control in Ch can be used as a learning tool for people interested in learning motion control.

5. Conclusions

This paper described the availability of Ch and its use as a motion control environment. Ch gives users the ability to control any motion system. It gives users an edge over standard applications since they are able to precisely control the system through either a C program run through the Ch interpreter or interactively through the prompt. It also allows the user to compile their software and form binary programs that can be optimized to run faster. Ch gives users considerable versatility and a quick prototyping capability. Utilizing Ch's ability for web control, users can create a web interface allowing web users to control mechatronic systems through the Internet. An example was given that showed the ease in integrating and controlling an IBM 7575 Robotic Manipulator through the Ch via the use of a National Instruments Motion Controller. The open source Ch NIMotion allows the user to control a multitude of axes at the same time interactively. The ideas and principles presented for open architecture interactive motion control using a C/C++ interpreter are applicable to any control board so long as it can be controlled using a C library.

Figure 4 Plots of position vs time and velocity vs time





References

Cheng, H.H. (1993), "Scientific computing in the Ch programming language", *Scientific Programming*, Vol. 2 No. 3, pp. 49-75.

Cheng, H.H. (1997), "Toward task-level robot programming", *Proc. 2nd Chinese World Congress on Intelligent Control and Intelligent Automation*, 23-27 June 1997, Vol. 1, pp. 648-53.

Cheng, H.H. (2001), "Real-time computing in open systems for manufacturing", *ASME Journal of Computing and Information Science in Engineering*, Vol. 1, pp. 1-8, 2001 March.

National Instruments (1999), *FlexMotion Software Reference Manual*, National Instruments Corp., Part Number 321943B-01, Firmware Version 6.9.3, August 1999 Edition.

SoftIntegration, Inc. (2004a), *The Ch Language Environment User's Guide*, available at: www.softintegration.com

SoftIntegration, Inc. (2004b), *The Ch Language Environment SDK User's Guide*,.

Wright, P.K. and Greenfeld, I. (1990), "Open architecture manufacturing: the impact of open-system computer on self-sustaining machinery and the machine tool industry", *Proc. ASME Manufacturing International*, March 1990, Vol. 2, pp. 41-7.