論文 92-29B-8-4

기능블록 구성에 의한 공정제어언어의 개발

(Development of a Process Control Language Using Function Block Configuration)

金 炳 國*

(Byung Kook Kim)

要 約

대규모 공정제어 시스템의 software개발을 용이하게 하고, 진보된 제어 알고리듬의 효율적인 구현을 위하여, 기능 블록 구성에 의한 공정제어언어를 개발하였다. 계층 구조의 다중 루프제어 시스템에 적합하도록 기능 블록 해석기 및 제어기의 2단계로 구현하였다. 장점으로는 실시간 제어기 파라메터 변경이 가능하고, 사용자 정의 기능 블록의 추가도 가능하며, 플랜트 모델 기능블록의 추가로 미리 제어성능을 입증할 수 있는점 등이다. Smith Predictor, auto tuner 등의 기능 블록을 구현하여 기능 블록 구성의 유용성을 입증하였다.

Abstract

A process control language is developed using function block configuration, to simplify software development for large scale process control systems, and to implement advanced control algorithms with ease. A function block parser and controller is implemented to be suitable for multi-loop control systems having hierarchical structure. On-line change of controller parameter is possible, and inclusion of user defined function block is also possible. By adding plant model block, control performance can be checked in advance. Function blocks of the Smith Predicotor, auto-tuners are implemented to demonstrate usefulness of function block configuration.

I. 서 론

반도체 및 컴퓨터 기술의 발달에 힘입어 공정제어 분야에 대규모 계층형 컴퓨터 공정제어시스템이 개발되어 외국에서는 널리 여러 공장에 설치 운용되고 있으나[5][7] 국내에서는 이러한 시스템이 개발단계에 머물러 있다.[3][4] 이러한 시스템은 제어의 분산화, 정보의 집중화를 통해 고신뢰도의 제어 성능과, 전체 공정의 효율적인 통합관제 제어기능이 매우 우수한 장점이 있다. 그러나 제어공학 측면에서 보면 sequence 제어와 연산기능과 PID제어가 주류를 이루고 있어서, 보다 진보된 제어 알고리듬을 적용하기가 곤란한 단점이 있다.

기능 블록 구성에 의한 공정제어 software의 구현은 공정제어 엔지니어가 특정한 공정에 대한 제어 시스템을 설계할때에 수월하게 software를 구현할 수 있게 하는 장점이 있다. 이 방법은 필요한 여러 제어기능 및 기타 기능들을 기능 블록으로 미리 구현하여 놓음으로써 추후 이들의 유기적인 결합으로 제어 software를 구현하는 방법으로 여러 공정제어 시스템에 사용되고 있다.

*正會員, 韓國科學技術院 電氣 및 電子工學科
(Dept. of Electrical Eng., KAIST)
接受日字 : 1992年 3月 17日

Taylor에서는 미리 표(table) 형식으로 정의된 다수의 기능블록들 중에서 필요한 블록을 선택하여 그 입출력을 연결하여 구성하는 방식(menu 선택방식)을 사용하였다. 문제점으로는 특정 기능별로 사용갯수의 상한선이 존재한다는 점이다.[7] Bailey에서는 기능블록 방식을 사용하였고, 임의의 갯수의 사용이 가능하다. 여기에는 sequence 및 여러 제어 블록들이 정의되어 있으나, 진보된 제어 블록은 결여되어 있다. 이 외에도 Honeywell, Yokogawa, Siemens 등 외국의 공정제어 시스템 생산회사들이 자사 나름대로의 제어 software 구성방식을 보유하고 있으며, 그들의 개략적 특성은 알 수 있으나, 구체적 know-how는알아내기가 어려운 실정이다. 국내의 공정제어software 구현에 대한 연구로써[8]에서 제안한 기능블록 방식은 입출력을 변수로 지정하였고, 입력과 parameter를 혼용하였다. 제약조건은 출력이 단 하나로 제한된다는 점과, 입출력 변수의 총 갯수, PID 블록등의 사용횟수에 상한선이 존재한다는 점이다. 또한[4]에서는 블록 사용횟수는 제한이 없으나, 출력이 하나로 제한된다는 점과, data structure가 2차원 array 로써 비효율적인 단점이 있다.

본 논문에서는 대규모 공정제어 시스템의 software 개발을 용이하게 하고, 진보된 제어 알고리듬의 효율적인 구현을 위하여 기능 블록 구성에 의한 공정제어 언어를 개발하였다. 기능 블록의 사용횟수에 제한이 없고, 입력과 parameter를 구분하며, 여러개의 출력을 허용하며, 효율적인 data structure를 생성하여 사용하는 장점이 있다.

계층 구조의 다중 루프제어 시스템에 적합하도록 기능 블록 해석기(parser) 및 제어기(controller) 를 구현하였다. 기능블록 해석기는 공정제어 엔지니어가 작성한 기능블록 구성 file을 읽어들여서 제어기가 사용할 수 있도록 data structure를 구성하는 역할을 한다. 제어기는 이 data structure를 이용하여 해당 기능 블록의 subroutine 프로그램을 매 sampling 시간마다 순차적으로 수행함으로써, 제어기능을 수행한다. 계층 구조의 상위에 존재하는 관리제어기(Supervisory 제어)에 의하여, 실시간으로 제어기 파라메터를 변경함이 가능하도록 하였다. 사용자가 새로운 기능의 기능블록을 정의 구현하여 추가함이 용이하도록 배려하였다.

4 차 이내의 시간지연을 포함한 플랜트 모델 기능블록의 추가로 workstation을 이용하여, 미리 전체 시스템을 가동하기 전에 제어성능을 확인하고, 제어 알고리듬의 수정이 가능하게 하여 제어 시스템 설계를 반복적으로 수행하여 만족스러운 제어성능을 얻을 수 있도록 하였다. 완성된 기능블록 구성은 공정제어 서브시스템에 전달되어 실시간 제어기능을 수행할 수 있게 된다. 기능블록 구성의 다양성, 확장성을 보이기 위하여 Smith Predictor 기능 블록을 구현하여 시간지연이 있는 시스템의 원활한 제어가 가능함을 보였으며, PID 계수의 자동 조정을 수행하는 auto tuner 기능 블록을 구현하여 그 유용성을 입증하였다.

본 논문은 다음과 같이 구성되어 있다. 2절에서는 적용대상인 공정제어 시스템의 hardware 구성을 설명하였고, 3절에서는 기능블록 구성, 4절에서는 기능블록 제어기에 대하여 설명하고 구현결과를 설명하였다. 5절의 결론으로 끝맺었다.

Ⅱ. Hardware 구성

계층구조의 대규모 공정제어 시스템은 제어 기능은 여러 제어 서브시스템에 분산시키고, 각종 공정정보는 집중시켜 중앙집중관리가 가능하게 시스템을 구성하는 방법으로 다음과 같은 서브시스템들로 구성되어 진다.

1. 관리제어 서브시스템(Supervisory Control Subsystem)

개발기능(Engineering Workstation) – 기능 블록 구성 화일의 편집, 사용자 기능코드의 개발

정보관리기능(Operator Interface Station) – graphic monitoring, 자료저장 기능

2. 공정제어 서브시스템(Process Control Subsystem)

실시간 제어기능 – Multi-loop 제어를 위한 primary 제어 모듈(PCM), back-up 제어 모듈(BCM), 신호 입출력을 위한 신호 I/O 모듈(SIM), 신호처리 및 단자 연결을 위한 신호 termination 모듈(STM), 자동/수동 절환모듈(AMS).

3. 시뮬레이션 서브시스템(Simulation Subsystem)

플랜트 시뮬레이션을 위한 CPU 모듈(PSM), 대상공정 simulation software.

이상과 같은 서브시스템들은 다음 그림 1과 같이 연결되어 전체 시스템을 구성한다.

Ⅲ. Software 구성

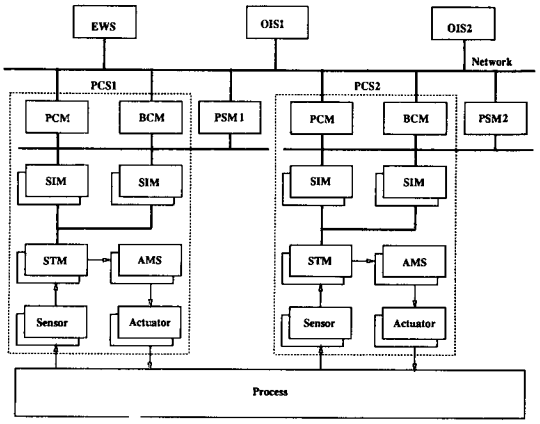

그림 1. 계층구조 공정제어 시스템
Fig. 1. Hierarchical process control system.

1. 기능블록 구성

대부분의 공정제어 형태는 여러개의 local loop의 복합적 구성으로 이루어진다. 경제적인 측면에서 다수의 single loop 제어기 보다는 이러한 복합적 loop을 일괄취급하는 제어기가 필요하게 되는데, 이것이 multi-loop 제어기이다. Multi-loop 제어기는 소형 경량, 연산 능력, 복합적 제어 기능 및 우수한 성능등이 요구된다. 제어기능으로는 여러개의 PID제어 기능 이외에 adaptive filtering, lead/lag 보상, 신호 처리 등의 다양한 기능을 내장해야 한다. 대규모의 공정의 제어를 위하여 기능의 모듈화에 의해 확장성을 갖고, 복잡한 제어 시스템의 구성이 가능하게 하여야 한다.

일반적으로 제어 시스템의 software 구현은 다음과 같은 단계들로 진행됨이 바람직하다.

1) 제어 알고리듬의 개발

시스템의 모델링, 제어 이론의 적용을 통한 안정되고 성능이 우수한 알고리듬을 workstation 또는 PC 등의 hardware와 상용의 software package (MatLab, Matrix-X 등)을 이용하여 개발 및 검증한다.

2) 제어 알고리듬의 coding 및 debugging

제어 software를 적절한 컴퓨터 언어로 coding하여 debugging 과정을 거쳐 완성한다. 이때 대상 공정에 대한 수학적 모델도 함께 프로그램되어야 폐 루프(closed-loop) 성능을 입증할 수 있다.

3) 제어기 구현 및 모의시험(Simulation)

공정제어 서브시스템에 제어 프로그램을 내장시키고, simulator와 결합하여 실시간(real-time) 성능을 확인한다.

4) 실제 process에 제어 시스템 구현

개발된 제어기와, process 출력을 측정할 수 있는 sensor 제어 출력을 process에 인가할 수 있는 actuator를 부착 연결하여 제어 시스템을 구현하고, commissioning 과정을 통하여 parameter 조정 및 제어 알고리듬 개선 등을 수행하여 제어 시스템을 완성한다.

본 논문에서는 위의 단계 2, 3 및 4에서 유용하게 사용될 수 있는 기능블록 구성 방식의 공정제어언어의 개발을 다루고 있다. 단계 2의 coding 과정은 보편적인인 high level의 언어를 사용하더라도 제어 알고리듬의 coding 및 debugging 에는 많은 시간과 노력이 소요된다. 이 프로그램 coding을 공정제어 엔지니어가 간편하게 할 수 있도록 기능블록 구성(function block configuration) 방식을 채택하였다. 이 방식은 제어 엔지니어에게 친숙한 block diagram으로 제어기를 구성할 수 있는 방식으로써 각각의 block에 해당하는 기능블록(function block)을 사용 조합하여 제어기를 구성하는 방법이다. 이 기능블록들은 제어 block diagram에서 각각의 block들에 해당하는 기능들로써, 자주 사용되고 유용한 기능들을 개발자가 function subroutine들로 미리 구현해놓고 사용자는 이들을 유기적으로 결합(configuration) 할 수 있게 하는 일종의 problem oriented language이다. 사용자는 각 기능블록 내의 구조를 알 필요가 없으며 단지 그 입력 출력 및 기능을 이해하면 된다. 이와 같이 단계 2, 3, 4를 효율적으로 수행할 수 있도록 해석기와 제어기 software가 workstion에서나 공정제어 서브시스템에서 모두 수행가능하도록 설계하였다.

제어기 구성은 그림 2와 같은 software 구조를 가지며 다음과 같은 단계로 구현된다.

1) 사용될 제어 알고리듬을 block diagram으로 표현한다.

2) 대상 process의 수학적 모델을 구한다.

3) Block diagram의 block들을 기능코드(function code)들로 표현한 configuration file을 작성한다. (Editor 이용)

4) Parser 프로그램을 수행시켜 data structure를 생성한다.

5) Controller 프로그램을 수행시켜 제어성능을 평가한다. 필요하면 3)부터 반복한다.

기능블록 구성화일의 문법(syntax)을 다음과 같이 정의하였다.

1) 하나의 제어기(Process Control Subsystem)가 하나의 구성화일(configuration file)을 이룬다.

2) 하나의 구성 화일은 여러개의 line으로 구성된다.

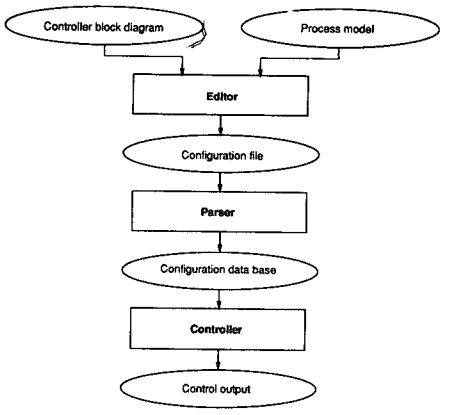

그림 2. 제어기 구성 software 구조
Fig. 2. Software structure of controller configuration.

3) 각 line은 하나의 기능 블록(function block)을 나타내며, 타 기능 블록과의 연결은 입출력 변수들로 이루어진다. 필요한 경우 기능 블록은 parameter들을 가질 수 있으며, 이들은 후일 on-line으로 수정 가능하다.

4) 각 line의 형식은 다음과 같다.

Output Function-name [inputs; parameters]

5) Output는 출력들을 나타내는 하나의 대표숫자로써 해당 기능블록의 블록번호를 나타내며, 수행순서를 나타낸다. Block 번호는 입출력 변수 array의 index를 나타내며, 출력이 여러개가 있는 기능 블록의 경우, 연속된 block 번호에 출력이 저장된다. Block 번호와 첫번째 출력 번호가 일치하므로 모든 Block은 하나 이상의 출력을 가져야 한다.

6) Block 번호가 작은 기능블록이 먼저 수행된다. 그리하여 몇개의 기능 block들이 loop을 형성하여 수행의 순서에 따라, 다른 결과를 유발할 수 있는 경우를 원천적으로 배제한다. Configuration engineer가 block 번호들을 설정하므로써 수행순서를 지정할 수 있게되며, 이는 후일 graphic configuration editor를 구현할 경우 필수적인 고려사항이 된다.

7) Function name은 알기쉽게 ADD, MAX, PID 등의 mnemonic을 사용한다. Function name은 임의의 길이의 영문자열(alphabetic string)로 구성되나 처음 6문자에 의해 다른 기능블록들과 구별되어야 한다. 각 기능블록은 특정한 갯수의 입력, 출력, parameter 갯수를 갖는다. 기호'['은 function name의 끝을 의미한다.

8) ']', ',', ',' ']' 등의 기호를 첨가하여 configuration의 출력, 입력, parameter들이 일목 요연하게 나타내진다. 또한 이러한 기호들을 check하여 parsing 수행시 error checking 기능을 수행하게 할 수 있다.

9) Inputs는 입력들을 나타내는 숫자들의 list로써 입출력 변수array의 index를 나타낸다. 기호 ';'은 입력들의 끝을 의미한다.

10) Parameters는 그 기능블록이 필요로 하는 parameter 값들의 list이며, 각 parameter들의 초기치들을 나타낸다. ']'은 parameter들의 끝을 의미한다. parameter들의 끝을 의미한다. Parameter에서 실수 값은 정수로 표시하여도 무방하다. 즉, 2.0인 실수의 parameter를 2로 표기하여도 무방하다.

11) 필요한 경우 기능 block은 internal parameter를 가질 수 있다. 이들은 기능블록내의 고유의 변수들로써, 후일 operator에 의한 변경이 불가능하다.

12) 빈 line이 허용되며, 한 line의 첫 글자가 '!'이면 comment line으로 처리한다.

예를들어 그림 3과 같은 간단한 PID 제어기는 다과 같이 기능 블록으로 구성될 수 있다.

Output	Name	Inputs	Parameters
1	ADJ	[;	0.5]
2	AIN	[;	1]
3	PID	[2 1 ;	2 10 0]
4	AOUT	[3 ;	2]
5	END	[:	]

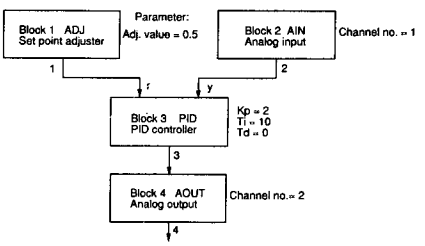

그림 3. PID 제어기의 기능 블록 구성
Fig. 3. Function block configuration of PID controller.

2. Data Structure

Configuration file에서 사용될 수 있는 기능코드들을 다음과 같은 data structure로써 정의하였다.

```
struct -fctype {
    int   index;          /*Sequential function code index*/
    char  fc[7];          /*Function name string*/
    int   fcn;            /*Function number     */
    int   no;             /*Number of outputs   */
    int   ni;             /*Number of inputs    */
    int   np;             /*Number of parameters*/
    int   nip;            /*Number of internal parameters*/
    int   ptype[MAX-PA]; /*Parameter types     */
} fct[MAX-FC];
```

각 기능블록에 대하여 위의 자료가 정의되어 있다. 또한 구현된 기능블록들의 parameter와 internal parameter들이 정의되어 있다.

Parsing 프로그램에 의해서 생성되는 data structure는 다음과 같다.

```
Struct -config {
        int         code;    /* Function code        */
        int         out;     /* Block no. (Output)   */
        int         *ip;     /* Input pointer        */
        UVAR        *pp;     /* Parameter pointer    */
} cf[MAX-BLK];
```

각 line의 기능 블록마다 cf[] 하나가 할당된다. Function subroutine의 출력은 var[] array에 integer 또는 float로 저장된다. Input는 var[] arry의 index만 input[ ] array에 integer로 저장하고 그 pointer는 cf[ ].ip에 저장한다. Parameter는 integer와 float를 혼합하여 사용하므로 UVAR pointer(union of integer, float)를 사용하여 par[ ] array에 저장하고 그 poniter를 cf[ ].pp에 저장한다. 또한 analog 입출력, digital 입출력, 입춘력 변수들이 global data로 shared memory에 저장된다.

3. 기능블록 해석기

Parsing 프로그램은 주어진 제어목적에 합당하게 작성된 configuration file을 하나 하나의 line별로 해석하여 제어기를 위한 data structure를 만들어 내는 것이 그 목적이다. 이 제어 data base는 제어기의 shared memory에 저장된다.

각 line을 읽어들여 먼저 빈 line이나 comment line은 무시한다. 다음으로 출력번호(block number)를읽어들인다. 아직은 해당 line의 기능코드가 무엇인지 모르므로 다만 하나의 수를 읽었는가를 확인한다. 다음에는 function name을 판별한다. 읽어들인 function name string을 우선 기능 번호로 변환한다. 기능번호는 long integer(32bit)으로써 위 두 bit는 0이고 다음의 각각의 5bit씩이 function name string의 ASCII 값의 하위(lower) 5bit를 포함한다. 그러므로 기능번호에는 function name string의 처음 6자의 정보만이 들어갈 수 있고, 이 처음 6자는 다른 function name과 구별될 수 있어야만 한다. 예를 들어 AIN은 Ox0297000, PID는 Ox2092000 등이다. 이렇게 변환하는 이유는 기능코드 table의 function name string들과 입력 function name과 string 비교들을 수행하는 것을 두 integer의 비교로 대체할 수 있어 비교시간이 훨씬 단축되기 때문이다.

기능코드 index를 찾으면 기능코드 table에서 해당 출력수, 입력수, parameter 수, internal parameter 수등의 정보를 읽어온다. 이 입력 수만큼 line에서 입력 변수를 읽어들여, input[ ] array에 저장하고, 그 첫째 입력의 index만을 cf[ ].ip에 저장한다. Parameter들은 별도의 par[ ] array에 저장한 후 첫째 parameter pointer 만을 cf[ ].pp에 저장한다. 필요한 경우 internal parameter들도 초기화한다.

각 기능블록마다 입력, parameter의 갯수가 다르지만 input[], par[] array에 빈틈없이 1차원 array로 저장하고, 각 변수는 입력 pointer, parameter pointer에 의하여 읽고 쓸 수 있다. 예를 들어 그림 3의 PID block의 parameter p는 par〔3〕에서 par〔5〕까지 저장되며 총 5개의 출력 변수, 3개의 입력 index, 6개의 parameter 변수가 소요되면 cf[ ] 5 개가 소요된다.

이상의 과정을 각 line에 대하여 반복하고, function name END가 나오면 parsing이 종료된다. 사용자의 configuration file에서의 실수 또는 오류를 발견하여 알려주는 오류검색기능으로 다음과 같은 사항을 검사 표시하도록 하였다.

1) 출력 index가 var[ ] array의 한계를 벗어난 경우 오류표시.

2) 전번 line에서 사용된 출력번호와 현재 line에서 사용된 출력번호간에 중복(충돌)이 있을 경우.

3) Function name이 기능코드 table에 없는 경우.

4) '[' 의 기호가 없을 경우.

5) ',' 의 기호가 없을 경우 또는 입력의 갯수가 틀릴 경우.

6) '[]' 의 기호가 없을 경우 또는 parameter의 갯수가 틀릴경우.

오류가 발생하면 해당 line은 무시하고 다음 line으로 넘어가서 검색을 반복하여 한번의 parsing에서 가급적 많은 오류를 발견할 수 있도록 하였다.

4. 기능 블록제어기

제어 및 simulation 프로그램은 prsing 프로그램에 의해서 생성된 data base를 받아들여 제어 및 process simulation을 수행하는 프로그램이다. 먼저 입력신호를 SIM에서 A/D변환기, digital 입력 등을 사용하여 받아들인다. Configuration data base를 한 item(line)씩 읽어서 해당 기능 subroutine을 부른다. 연산 결과는 출력 변수 array에 적절히 저장되어 후일 다른 기능 subroutine부터 수행하여 END function이 나올때까지 수행하며, 출력신호를 SIM의 A/D 변환기, digital 출력으로 내보낸다. 그 이후는 다음 sampling time까지 기다린다.

Simulation 기능을 위하여 그림 4와 같이 DATOV (DA to variable), VTOAD(variable to A/D) 등의 기능 블록이 추가로 구현되었다.

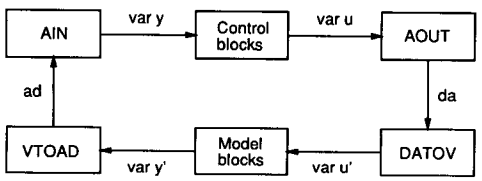

그림 4. Simulation용 기능 블록 구성
Fig. 4. Function block configuration for simulation.

5. On-Line Parameter 변경

Commissioning 과정등에서 기능블록 내부의 paramenter를 변경시킬 필요가 있으므로, Enginering Workstation이나 Operator Interface Station 에서 parameter 변경에 대한 명령을 받아들인다. 명령은 다음과 같은 형태로 정의하였다.

Change PCS number, block number, parameter index, parameter value

이 명령은 network를 통하여 해당 PCS에 전달되어, 해당 block의 index번째의 parameter 값을 새로운 값으로 대체하게 된다.

Ⅳ. 제어기능 블록

다음 표 1과 같이 기능블록이 현재 정의 및 구현되었으며, 이는 공정제어에서 많이 쓰이는 대표적인 것들이며, 추후 추가 또는 삭제가 가능하다. C 언어를 사용하여 기능 subroutine 형태로 구현하였다.

1. PID제어 블록

교과서의 PID 제어는 다음의 형태를 갖는다.

$$u(t)=K_P[e(t)+\frac{1}{T_i}\int_0^t e(t)\,dt+T_d\frac{de(t)}{dt}]$$

3개의 parameter는 proportional gain K_P, integral tim T_i, derivative time T_d라 부른다.

미분항의 목적은 폐 루프 안정도를 향상시키는데 있다. 그러나 높은 주파수의 noise가 크게 증폭되는 문제점이 발생되므로 고주파 측정 noise가 최대 N배로 증폭되도록 다음과 같은 modified derivative term을 사용한다.

$$D=-\frac{s\ K_P\ T_d}{1+s\ T_d/N}y$$

미분항이 출력 y만의 함수임에 유의하여야 한다.

Integrator가 지속적이고 큰 오차가 발생할 경우 그 출력이 곧 actuator의 동작범위를 벗어나게 되는 바, 이에 의한 큰 response overshoot가 발생하게 되는데 이를 integrator windup이라 한다. 이를 방지하기 위하여 anti-reset windup(ARW) 방식이 필요하게 된다. 한가지 ARW 방법은 다음과 같다.

$$I=\frac{K_P}{T_i}\int_0^t(r-y)\,dt+\frac{1}{T_t}\int_0^t(u-v)\,dt$$

$$u=P+I+D$$

$$u=sat(v)$$

여기서 T_t는 tracking시정수라고 부른다. 실제제어 입력 u(saturated)와 computed (non-saturated) 제어 입력 v의 차이를 T_t의 시정수로 integral 항을 보정하여 줌으로써 integrator wind-up을 방지할 수 있다.[2]

비례항을

$$P=K_P(b\cdot r-y)$$

로 설정함으로써, load disturbance 및 설정치 변화에 대하여 모두 좋은 응답을 얻을 수 있다. 그리하여 실적인 PID제어 기능은 총 6개의 parameter-K_P, T_i, T_d, T_i, T_d, T_t, N, b-를 갖는다.

이 PID 제어기를 discretize화 하는 방법은 다음과 같다. 비례항은 간단히

$$p(t_k)=K_P[b_r(t_k)-y(t_k)]$$

와 같은 변환된다. 적분항은 다음과 같은 recursive form으로 나타내진다.

$$I(t_k+1)=I(t_k)+\frac{K_P*h}{T_i}e(t_k)$$

여기서 h는 sampling time을 나타낸다. 미분항은 T_d의 모든 값에 대하여 적은 오차를 갖는 Tustin의 근사적 방법을 사용하여, 다음과 같이 discretize한다.

$$D(t_k)=\frac{2\,T_d-hN}{2\,T_d+hN}D(t_k-1)-$$

$$\frac{2KNT_d}{2\,T_d+hN}[y(t_k)-y(t_k-1)]$$

2. Model 기능 블록

Model 블록을 구현하여 폐루프 시뮬레이션이 가능하도록 하였다. Model은 한블록당 최대 4차까지의

표 1. 기능 블록의 정의
Table 1. Definition of function block.

ID	블록이름	출력수	입력수	Param수	Internal수	내 용
Ⅰ.	Process I/O Functions					
1	AIN	1	0	1	0	Analog input
2	AOUT	1	1	1	0	Analog output
3	DIN	1	0	1	0	Digital input
4	DOUT	1	1	1	0	Digital output
5	ADJ	1	0	1	0	Adjuster input
6	SW	1	0	1	0	Switch input
Ⅱ.	Arithmetic Functions					
10	ADD 4	1	4	0	0	Add 4 variables
11	ADD	1	2	2	0	Add 2 variables with scale
12	MUL	1	2	1	0	Multiply 2 variables
13	DIV	1	2	1	0	Divide 2 variables
14	AVE	1	2	1	0	Average of 2 variables
15	ABS	1	1	2	0	Absolute value
16	FEXP	1	1	1	0	Exponetial value
17	FLOG	1	1	1	0	Natural logarithm
18	ROT	1	1	0	0	Square root
Ⅲ.	Selector Functions					
20	MAX	1	4	0	0	Maximum of 4 variables
21	MAX	1	4	0	0	Minimum of 2 variables
22	LIM	1	1	2	0	Upper and lower limit
23	RATE	1	1	2	1	Rate limit
24	FG	1	1	12	0	Function Generator
25	ASEL	1	3	0	0	Analog selector
26	DSEL	1	3	0	0	Digital selector
Ⅳ.	Control Functions					
30	DELAY	1	1	1	256	Dead time(time delay)
31	LDLG	1	2	2	5	Lead lag
32	PID	1	2	6	3	PID control
33	PIDE	1	1	6	3	PID control with error
34	AMS	1	1	2	0	Auto Manual Station
35	ZTUNE	2	3	3	12	Ziegler Nichols auto-tuner
36	RTUNE	2	3	3	18	Relay auto-tuner
37	FILTER	1	2	1	5	Filter
Ⅴ.	Process Model					
40	MODEL	1	1	Ⅲ	522	Process model
41	SMITH	1	1	11	782	Smith Predictor
58	VTOAD	1	1	1	0	Var. to A/D
59	DATOV	1	0	1	0	D/A to var.
Ⅶ.	User defined function block					
90	USER0	1+	1+	1+	1+	+사용자 설정
....	.....	.....	.....	.....		
99	USER9	1+	1+	1+	1+	

공정 모델을 simulate할 수 있다. 5차 이상은 cascade 형태로 구현할 수 있다. 여기에 정상상태 이득 K, 입력 시간지연 d를 다음과 같이 아울러 설정할 수 있다.

$$G(s)=\frac{K\ exp(-d\ s)}{1+a1*s+a2*s^2+a3*s^3+a4*s^4}$$

시간지연을 수용하기 위하여는 과거의 입력값들을 저장할 필요가 있는데 이를 위하여 circular buffer를 사용하였다.

모델 동특성의 보다 정확한 계산을 위하여, 4차의 Runge Kutta integration 방식을 채택하여 구현 하였

다. 이 계산을 위하여 plant dynamics를 state space observable canonical form으로 나타내었고, state matrix계 계수를 초기확 하기 위하여 init model routine을 프로그램하였다. 출력값도 후일 참조를 위하여 circular buffer에 저장하였다.

3. Smith Predictor 블록

Process의 time delay가 비교적 큰경우 PID제어는 unstable하거나, slow response를 나타낸다. 이 경우 Smith Predictor P(s)를 사용하면 좋은 response 를 얻을 수 있다.

$$P(s)=\frac{K[1-exp(-d\ s)]}{1+a1*s+a2*s^2+a3*s^3+a4*s^4}$$

적용상의 선결조건은 plant dynamics를 정확히 알아야 한다는 점이다. 이 Smith Predictor 제어에 대한 기능 블록 구성은 그림 5 및 다음과 같다.

```
 1 ADJ      [;     1.0]
 2 AIN      [;     1]
 3 ADD      [2 5;  1 1]
 4 PID      [3 1;  10 5.0 0  1 3 5]  3.
 5 SMITH    [4;    11.55 1.0 9.8 0 0 0  0 0 0 0]
 6 SMITH    [4;    1]
 7 AOUT     [5 ;   11.55 1.0 9.8 0 0 0  0 0 0 0]
 8 MODEL    [7;    1]
 9 DATOV    [ ;    1]
10 END      [ ;    1]
```

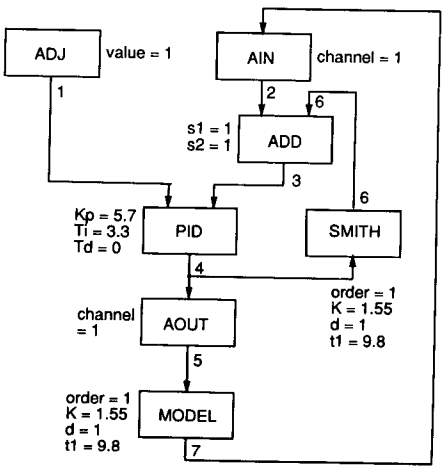

그림 5. Smith Predictor 기능 블록 구성
Fig. 5. Function block configuration of the smith predictor.

제어 결과를 그림 6에 나타내었다. Response가 시간 지연 1초만큼 delay된것을 제외하고는 delay 없는 plant와 동일한 응답을 나타냄을 알 수 있다. 결과의 효율적인 plot을 위하여 X window protocol을 사용하였다.[9,10]

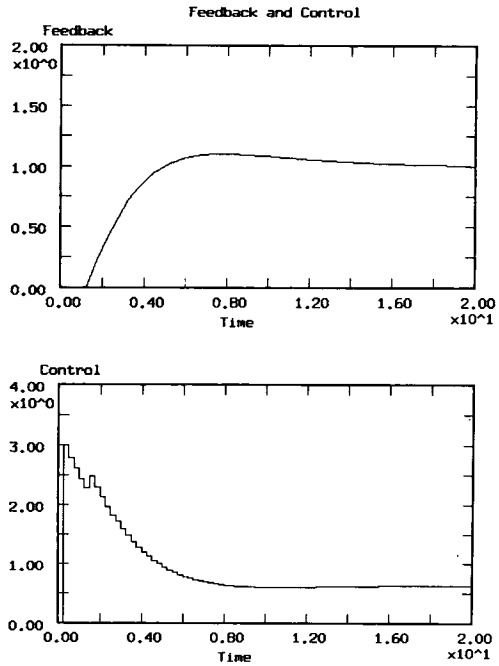

그림 6. Smith Predictor의 응답
Fig. 6. Response of the smith predictor.

4. Auto-Tuner 블록

공정제어 시스템에서 가장 널리 사용되는 PID 제어기의 PID이득값을 tuning하는것이 operator의 중요한 임무중의 하 나이다. 그러나 이 작업은 지루하고, 주기적으로 경년 변화에 대응하여 시행하여야 하므로 제대로 tuning 되지 못하는 경우가 적지 않다. 이러한 tuning 작업을 operator의 요구 또는 외부 신호에 의해 자동적으로 수행하게 하는것이 auto-tuning의 목적이다. 본 연구에서는 이러한 auto-tuning을 기능블록 구성 방식에서 사용할 수 있도록 구현하였다.

지금까지 알려진 auto-tuning 방법중에서 널리 알려진 방법으로 자동화가 가능한 방법으로 Ziegler Nicho의 step response method와 Astrom의 relay method를 들 수 있다.[1,2] Ziegler Nichols의 frequency response 방법은 시스템이 critically stable할때까지 제어이득을 증가시키는 방법으로 실제로 컴퓨터에 의해 공정에 적용 하기에는 안전성 등의 문제가 있다.

Ziegler Nichols의 step response method는 시스템의 open-loop response를 이용하는 방법으로 step response의 최대 slope인 점을 찾아내고 이점에서의 접선과 시간축이 만나는 점을 창아서, system lag L과 최대 slope R을 결정한다. 이 두 parameter로부터 PID parameter-proportional gain K, integral time Ti, derivative time Td-를 표 2와 같이 결정한다.

표 2. Ziegler Nichols step response방법의 PID 계수

Table 2. PID Coefficiends of ziegler Nichols step response.

Controller	K	Ti	Td
P	1/RL	-	-
PI	0.9/RL	3L	-
PID	1.2/RL	2L	0.5L

이 tuning 방법은 load disturbance에 좋은 특성을 가지고 있으나, step 응답은 overshoot가 꽤 커서, 폐루프 시스템의 응답은 1/4 damping의 특성을 가지고 있음이 알려져 있다. 이에 해당하는 damping factor는 0.22이다.

이러한 방법으로 auto-tuner를 그림 7과 같이 구성할 수 있다.

```
 1 SW     [;        1]
 2 ADJ    [;        1.0]
 3 AIN    [;        0]
 4 ZTUNE  [3 2 1 ; 0.1 10.0  5]
 5 PID    [3 2 ;   3.0 5.0 0.0  1 3 5]
 7 ASEL   [6 4 5 ;]
 8 AOUT   [7 ;      0]
 9 MODEL  [11;     2 2.0  0 10.0 25.0  0 0   0·0 0 0]
10 VTOAD  [9 ;      0]
11 ADTOV  [;        0]
99 END    [;        ]
```

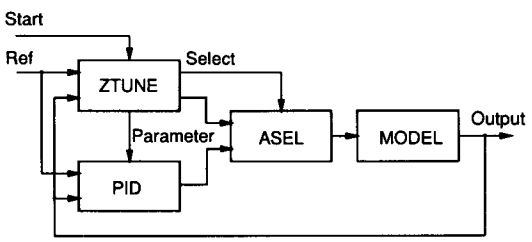

그림 7. Auto-tuner의 기능 블록 구성
Fig. 7. Function code configuration of auto-tuner.

Auto-tuning을 시작하기 전에 plant가 정상상태에 도달해 있다고 가정한다. Start 입력에 의해 auto-tuning이 시작되면, 정상상태 제어출력 u에 크기 m인 step을 더하여, 출력으로 내보내면, ASEL block에 의해 plant에 입력이 인가된다. 시간 T동안 step response를 측정하고, lag, slope를 검출해 낸다. 또 시간 T동안 정상상태 제어 출력만을 plant에 인가하여, step down response를 측정하여 또 lag, slope를 측정해 낸다. 두 측정치의 평균을 최종 lag, slope로 한다. 이를 토대로 PID 계수를 위표에 의해 계산하여, 해당 PID block의 계수로 설정한다. 총2T의 시간이 소요된다. ZTUNE의 prrameter는 제어입력의 크기, tuning time T, 해당 PID block 번호이다. 그림 8에 수행 결과를 실었다.

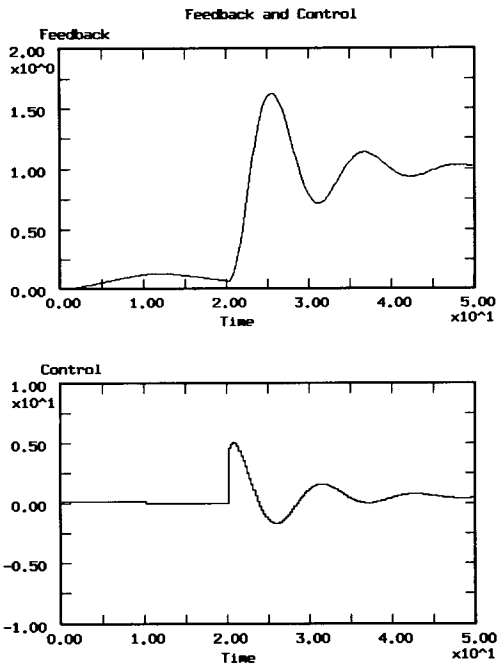

그림 8. Ziegler Nichols auto-tuner 응답
Fig. 8. Response of Ziegler Nichols auto-tuner.

두번째 auto-tuning 방법으로 Astrom의 relay method를 사용한 auto tuner 블록 RTUNE을 구현하였다. Relay method는 relay feedback을 이용하는 방법으로, approximate oscillation condition을 얻는 방법이다. 인가한 relay의 출력을 d, 출력의 oscillation의 크기를 a라 하면, 임계 이득 kc는 $\pi a/4d$로 주어진다. 출력의 주기로 임계주기 tc를 구한다. 여기서 표3

표 3. 주파수응답 방법의 PID계수
Table 3. PID Coefficiends of ziegler Nichols stdp response.

Controller	K	Ti	Td
P	0.5kc	-	-
PI	0.4kc	0.8tc	-
PID	0.6kc	0.5tc	0.125tc

에 의해서 PID gain parameter를 구한다.

Start 입력에 의해 auto-tuning이 시작되면, relay 제어가 3회 반전될때까지 tuning을 수행한다. 3회의 평균 ultimate gain kc, ultimate period tc를 계산하여, 이를 토대로 PID 계수를 위 표에 의해 계산하여, 해당 PID block의 계수로 설정한다. 총 소요시간은 plant에 따라 달라진다. Relay에 hysteresis를 첨가하면, noise에 의한 측정의 영향이 감소되고, oscillation period는 늘어난다. 기능 블록 구성은 그림 7에서 ZTUNE을 RTUNE으로 대체한것과 동일하다. RTUNE의 parameter는 relay의 magnitude, hysterisis, 해당 block의 번호이다. 그림 9에 수행 결과를 실었다.

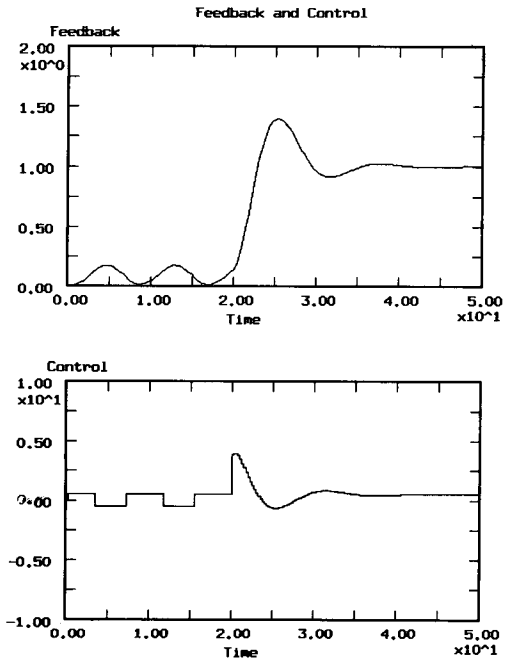

그림 9. Relay auto-tuner 응답
Fig. 9. Response of relay auto-tuner.

5. 사용자 정의 기능블록

사용자 정의 기능 블록을 작성하려면 먼저 기능코드 table에 기능 블록의 이름(USERn을 대신할 이름) 출력, 입력, parameter, internal parameter의 갯수, parameter의 type(integer/float)를 정의한다. 기능 블록을 C function subroutine으로 작성한다. parameter의 초기화가 필요한 경우 초기화 routine도 아울러 작성하고, compile하여 link시키면 된다.

V. 결 론

대규모 공정제어 시스템의 software로써, 기능 블록 구성에 의한 공정제어 언어를 기능 블록 해석기(parser) 및 제어기(controller)로 구현하였다. 이는 제어이론과 공정제어의 교량 역할을 수행하는 engineeering software로써, 다음과 같은 장점이 있다. 실시간 제어기 파라메터 변경이 가능하고, 사용자 정의 기능 블록의 추가도 가능하여, 진보된 제어블록의 구현이 용이하다. 플랜트 모델 기능블록의 추가로 미리 제어 성능을 입증할 수 있도록 하였다. 기능블록의 사용횟수에 제한이 없고 여러개의 출력을 허용하며, 입력과 parameter를 1차원 array에 저장하여, 효율적인 data structure를 생성하여 사용할 수 있다.

적용예로써, Smith Predictor, auto tuner 등의 기능 블록을 구현하여, 그 유용성을 입증하였다. 본 공정제어 언어를 토대로하여 이미 개발된 수많은 진보된 제어 알고리듬을 공정현장에 용이하게 적용함이 국내기술로도 가능하게 되었다.

추후 연구과제로는 적응제어 등의 다양한 진보된 제어 알고리듬의 기능 블록화를 통하여 실제 시스템에 적용이 용이하게 하는 것. 고신뢰도 제어 시스템에 대한 연구등이 있고, 궁극적으로 신뢰성, 융통성, 확장성이 있는 디지탈 계장 제어 시스템의 국산화, 표준화, 고성능확로써 제어 기술의 고도화에 기여하는 것을 목표로 한다.

參 考 文 獻

[1] K.J. Astrom, B. Wittenmark, "Adaptive control", Addison Wesley Publishing, 1989.
[2] K.J. Astrom, T. Hagglund, "Automatic Tuning of PID controllers", Instrument Society of America, 1988.
[3] "마이크로 컴퓨터를 이용한 전자제어 시스템의 고신뢰화에 대한 연구,", 한국전력공사 기술연구원, KRC-881-J03, 1990. 8.

[4] "발전소 제어용 디지탈 계장제어 시스템 개발", 한국전력공사 기술연구원, KRC-991-J03, 1990. 8.
[5] Bailey controls, "Bailey Network 90".
[6] Bailey controls, "Function code application manual".
[7] Taylor Instruments, "MOD300 process control system".
[8] 정현규, "제어 언어를 이용한 다중루프 프로그램형 제어기에 관한 연구", 한국과 한국과학기술원 석사학위 논문 , 1988.
[9] MIT, "X window systems-Version 11 Release 4", MIT.
[10] E. Johnson, K. Reichard, "X window applications programming," Management Information Source, Inc., 1989

著 者 紹 介

金 炳 國 (正會員)
1952年 10月 5日生. 1975年 서울대학교 전자공학과 졸업. 1991년 한국과학기술원 전기 및 전자공학과 공학박사. 1981~1986年 우진계기(주) 연구실장. 1982~1984年 University of Michigan방문연구 1986~현재 한국과학기술원 전기및 전자공학과부교수. 주관심분야는 로보틱스, 컴퓨터 공정제어, 고신뢰도 제어, 지능 제어등임.