developer Works

MBD 기반의 임베디드 시스템 개발

자동 코드 생성 툴의 활용

지금까지는 시뮬레이션을 위해서 플랜트와 제어기를 모델링했다. 시뮬레이션은 말 그대로 PC 상에서 테스트를 수행하는 것이므로 지금부터는 실제의 실시간 시스템에서 제어기의 성능을 테스트해 보도록 한다. 우선 이 테스트에 필요한 자동 코드 생성 툴 등을 살펴보도록 하자.

4

이영준 cipher@cipher.pe.kr | 어떠한 '삽질'도 하지 않고 가능한 많이 게으름을 피우고자 툴을 최대한 활용하고 있는 게으른 엔지니어다. 언젠가는 임베디드 소프트웨어의 아키텍처 설계자가 되길 희망한다. 하지만 그러기 위해서는 알아야 할 게 너무 많고 또한 게으름을 피워선 안 되므로 이에 대해 늘 고민하며 괴로워하고 있다.

시뮬레이션으로 제어기 설계를 마쳤으면 이제는 실시간으로 성능을 테스트해야 할 시간이다. 실시간으로 성능을 테스트하기 위해서는 먼저 실시간으로 테스트하는 데 필요한 C 코드가 있어야 한다.

자동 코드 생성

C++의 경우에도 임베딩하는 경우가 있지만, 여전히 C 언어가 임베디드 소프트웨어로 가장 많이 사용되고 있다. 제어기나 복잡한 알고리즘을 시뮬레이션 함으로써 테스트를 끝내고 이를 바탕으로 다시 처음부터 C 코드를 작성하게 되면 엄청난 시간과 노력이 들게 된다. 여기서 이럴 때 사용할 수 있는 코드 생성에 대해 살펴보자.

자동 코드 생성의 유용성과 효율성

자동 코드 생성 툴의 유용성에 대해 한 가지 예를 들어 설명해 보자. Flight Code라는 용어는 항공기에서 비행을 위해 작성하게 되는 모든 소스 코드를 일컫는 말이다.

국방 항공 분야의 강자인 록히드 마틴(Lockheed-Martin)에 따르면 이런 Flight Code는 1,500만 LOC(Line Of Code)를 넘는다고 한다. 국방이나 항공 분야에서 문제가 발생하면 크나큰 인명 손실을 초래할 수도 있으므로 소스 코드를 작성할 때는 수없이 많은 테스트를 거쳐야 한다. 그리고 개발 단계에서는 아직 하드웨어가 완성되어 있지 않으므로 시뮬레이션을 통해 이런 작업을 하게 된다. 이때 자동 코드 생성 툴을 사용하지 않는다면 시뮬레이션으로 테스트한 알고리즘을 다시 처음부터 C로 작성하는 것이므로 시간과 비용의 소모가 엄청나게 된다. 때문에 자동 코드 생성 툴을 쓰게 되면 이런 C 코드 생성에 필요한 시간과 비용을 절약하게 된다.

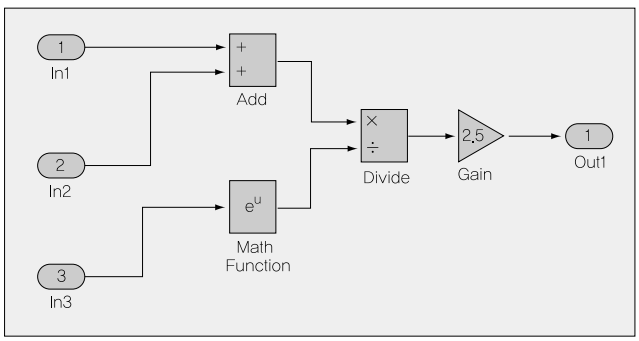
SAE(Society of Automotive Engineers)는 자동차나 비행기처럼 엔진 등의 힘에 의해 움직이는 것들을 취급하는 엔지니어들을 위한 가장 큰 협회 가운데 하나이다. 2004년에 Visteon이라는 회사가 자동 생성된 코드와 직접 손으로 작성한 코드를 비교한 논문을 제출했는데, 여기서는 LOC 비교가 아니라 생성된 코드를 위해 필요한 ROM과 RAM의 사이즈를 비교했다. 〈표 1〉에 그 결과가 나와 있듯이 손으로 작성한 코드보다 자동 생성된 코드가 나은 결과를 나타냄을 알 수 있다. 자동 코드 생성 툴을 이용한 결과를 보면 실제로 직접 작성한 코드와 비슷하거나 혹은 조금 떨어지는 결과를 보여주는 것이 일반적이지만, 사용되는 비용과 시간을 생각해 보면 자동 코드 생성 툴을 충분히 이용할 만한 것으로 평가할 수 있다.

	Hand Code	Auto Code
ROM	6408	6192
RAM	132	112

〈표 1〉 직접 작성한 코드와 자동 생성된 코드 비교

자동 코드 생성 툴

Simulink 모델에 대한 자동 코드 생성 툴의 대표적인 소프트웨어로 같은 회사(MathWorks)의 제품인 Real-TIme Workshop(이하 RTW)을 꼽을 수 있다. 또 다른 소프트웨어로는 dSpace사의 TargetLink가 있다. 지난해부터 MathWorks사는 소프트웨어를 6개월에 한 번씩 출시하고 있는데 RTW의 경우 Simulink의 버전이 변경됨과 동시에 변경된 버전을 지원하고 있다. 하지만 TargetLink의 경우는 Simulink 버전의 변화를 함께 따라가지 못하고 지난 버전만을 지원하고 있다. 아울러 TargetLink는 생성되는 코드를 사용자가 원하는 형태로 어느 정도 수정할 수 있는 기능을 갖춘데 반해, RTW의 경우는 이런 사용자 수정을 할 수 없게 되어 있다. 따라서 이런 작업을 하기 위해서는 Real-Time Workshop Embedded Coder(이하 RTW-EC)를 따로 구매해야 한다. 이 글에서는 필자가 익숙한 RTW와 RTW-EC를 이용해 설명하도록 하겠다.

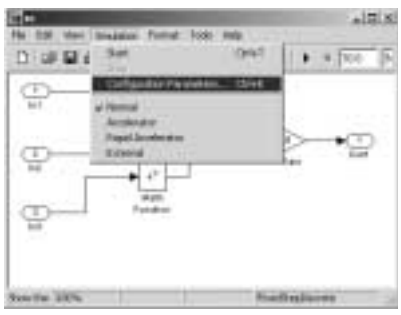


〈그림 1〉 간단한 사칙 연산을 하는 Simulink 모델

복잡한 모델을 이용해 설명하면 전체 코드를 보여 주지 못하므로 〈그림 1〉처럼 간단한 사칙 연산을 하는 모델을 만들고, 이후 RTW와 RTW-EC로 코드를 생성해 어떤 차이가 있는지를 확인해 보자.

Real-Time Workshop

RTW를 사용하기 위해 〈화면 1〉과 같이 선택하면 〈화면 2〉와 같은 윈도우가 나타난다. 화살표로 설명한 부분 가운데 System target file 부분에 있는 것은 grt.tlc 파일이 코드가 어떻게 생성되기를 바라는지에 대한 내용을 미리 세팅한 것이다. 왼쪽에 있는 메뉴에서 Real-Time Workshop 메뉴 아래에 생성되어 있는 메뉴는 System target file을 선택함에 따라 다르게 나타날 수 있다. RTW-EC의 경우 더 많은 메뉴가 생성된다.



〈화면 1〉 Configuration Parameter 선택 메뉴

〈화면 2〉 RTW 메뉴

〈화면 3〉 생성된 코드를 보기 위한 HTML 리포트

이제 Generate code 버튼을 클릭하면 〈화면 3〉과 같은 윈도우가 생성되면서 생성된 코드를 볼 수 있게 해준다. 코드를 생성하고자 하는 모델의 이름이 sc.mdl 파일이므로 sc.c와 같이 모델의 이름이 붙은 파일들이 생성된다. 소스 코드는 sc_grt_rtw라는 폴더 안에 포함되며 html 리포트는 html 디렉토리 밑에 sc_codegen_rpt.html 파일을 열어 확인할 수 있다. 이를 통해 sc.c 파일에 실제로 모델 알고리즘이 들어 있는 함수가 있으며 함수의 이름을 어느 정도 짐작할 수 있을 것이다. 리포트 윈도우에서 함수에 관련된 부분만 보게 되면 〈리스트 1〉과 같다. 예상한 것처럼 함수의 이름은 sc_output이다. 생성된 코드에서 주석 부분은 Simulink 모델 중에서 어떤 블럭과 연계되어 있는지를 보여준다. 물론 옵션으로 이런 주석 부분을 없앨 수도 있다.

〈리스트 1〉 리포트 윈도우에서의 함수 관련 부분

```
27  /* Model output function */
28  static void sc_output(int_T tid)
29  {
30    /* Sum: '<Root>/Add' incorporates:
31     *  Inport: '<Root>/In1'
32     *  Inport: '<Root>/In2'
33     */
34    sc_B.Add = sc_U.In1 + sc_U.In2;
35
36    /* Math: '<Root>/Math Function' incorporates:
37     *  Inport: '<Root>/In3'
38     */
```

```
39     sc_B.MathFunction = exp(sc_U.In3);
40
41     /* Product: '<Root>/Divide' */
42     sc_B.Divide = sc_B.Add / sc_B.MathFunction;
43
44     /* Gain: '<Root>/Gain' */
45     sc_B.Gain = sc_P.Gain_Gain * sc_B.Divide;
46
47     /* Outport: '<Root>/Out1' */
48     sc_Y.Out1 = sc_B.Gain;
49     UNUSED_PARAMETER(tid);
50   }
```

모델에서 코드를 생성하게 되면 옵션을 선택해서 간단하게 만들 수 있는 코드는 최대한 간단하게 만들 수 있다. 〈리스트 2〉의 코드는 주석문을 넣지 않고, Block Reduction과 Signal Reuse 등의 옵션을 체크해 나온 결과를 보여준다.

〈리스트 2〉 옵션 체크 결과

```
8   static void sc_output(int_T tid)
9   {
10     sc_Y.Out1 = (sc_U.In1 + sc_U.In2) / exp(sc_U.In3) *
sc_P.Gain_Gain;
11     UNUSED_PARAMETER(tid);
12  }
```

Real-Time Workshop Embedded Coder

지금까지는 RTW를 이용해 생성된 코드를 보여줬다. 실제로 RTW는 간단히 말해 Hardware-In-the-Loop(HIL)나 Rapid Control Prototyping을 위한 코드를 생성해 준다. 실제로 임베딩하기 위해 코드를 생성해야 하면 RTW-EC를 이용해야 한다. RTW-EC는 이름이 의미하는 것처럼 임베딩되기 위한 최적화된 코드를 생성해 준다. 〈표 2〉는 〈그림 1〉에 있는 모델을 RTW와 RTW-EC로 코드를 생성했을 때 생기는 파일들과 LOC이다. rtwtypes.h 파일이 RTW-EC에서 더 많은 LOC를 보여 주는 것은 RTW에서 생성된 파일은 다른 헤더 파일을 인클루드(include)하고 있기 때문이다. RTW-EC에서 생성된 파일은 다른 파일을 인클루드하더라도 ANSI C 헤더 파일만을 인클루드하기 때문에 LOC가 커지는 것이다.

Files	RTW Code	RTW-EC Code
ert_main.c	x	o(29)
rt_nonfinite.c	o(142)	x
sc.c	o(205)	o(24)
sc_data.c	o(6)	o(6)
rt_nonfinite.h	o(19)	x
rtmodel.h	o(5)	x
rtwtypes.h	o(21)	o(171)
sc.h	o(866)	o(47)
sc_private.h	o(25)	o(17)
sc_types.h	o(7)	o(8)

〈표 2〉 RTW와 RTW-EC에서 생성된 파일과 LOC

RTW-EC는 기본적으로 다른 시스템에서 C 코드와 합쳐 실행되는 것을 목표로 하는 코드를 생성하므로 인풋 인자를 가지는 함수 형태로 만들 수 있다. 〈리스트 3〉의 코드는 인풋 3개와 아웃풋 1개를 가지고 게인 값을 변경시킬 수 있도록 함수의 인자(총 5개)로 받을 수 있게 생성된 함수이다. 이와 같이 함수 형태로 만들어서 다른 환경, 즉 임베딩시키고자 하는 환경에서 사용할 수 있도록 만들어 주는 제품이 RTW-EC이다.

〈리스트 3〉 5개의 인자를 가진 함수

```
4   void sc_step(Parameters_sc *sc_P, real_T sc_U_In1,
real_T sc_U_In2, real_T
5                sc_U_In3, real_T *sc_Y_Out1)
6   {
7     (*sc_Y_Out1) = (sc_U_In1 + sc_U_In2) / exp(sc_U_In3)
* sc_P->Gain_Gain;
8   }
```

그러면 RTW-EC로 생성된 코드는 얼마나 정확하고 효율적일지가 궁금해진다. 이는 이미 수많은 프로젝트에 사용되어 그 효율성과 정확성은 어느 정도 입증되었다고 본다.

시뮬레이션과 실시간 테스트

이제는 실제 하드웨어에서 실시간으로 플랜트를 제어할 때도 같은 성능이 나오는지를 테스트해야 한다. 이미 알고 있듯이 실시간으로 제어하기 위해서는 윈도우XP와 같은 범용 운영체제를 사용하지 않고, 실시간 성능을 지원하는 운영체제를 써야 한다. 가장 많이 쓰는 실시간 운영체제는 윈드리버의 VxWorks이고, 그 외에 리눅스에서 실시간 성능을 구현한 MontaVista와 교육용으로 많이 쓰는 uC-OSII 등이 있어 선택의 폭을 넓혀준다.

범용 운영체제와 다르게 이런 실시간 운영체제는 특정한 보드에 포팅하거나 혹은 입출력을 위한 보드에 대한 드라이버를 직접 작성해야 하는 수고가 따른다. 이런 과정은 하드웨어 종속적인 작업인 탓에 항상 수작업이 요구된다. 시뮬레이션으로 제어기 설계를 완료하고 이제 실시간으로 성능을 테스트하기 위해 다시 수작업으로 이런 일들을 해야 하므로 역시나 시간 소모적인 작업이라 할 수 있다.

그럼 Simulink로 만든 모델을 실시간으로 빠르게 테스트해보고 싶다면 어떻게 해야 할까? 여기서 바로 Rapid Control Prototyping(RCP)이 등장하는 것이다.

Rapid Control Prototyping

RCP는 엔지니어가 시뮬레이션을 통해 테스트한 알고리즘을 쉽고 빠르게 실제 하드웨어와 연결함으로써 실시간으로 알고리즘을 테스트할 수 있는 방법을 말한다. 여기서 '쉽고 빠르게'가 강조되는 것에서 알 수 있듯이, 시뮬레이션에 사용한 모델에 대해 그대로 자동 코드 생성 툴을 써서 코드를 생성하고, 이어서 실시간 운영체제와 IO를 위한 디바이스 드라이버 등을 모두 제공받아 사용하는 것이다. 즉 엔지니어가 개발한 알고리즘을 다시 C로 코딩하는 작업과 실시간으로 하드웨어를 돌리기 위한 실시간 운영체제의 포팅과 디바이스 드라이버 작업을 하지 않고 하드웨어의 연결로 빠르고 쉽게 알고리즘을 테스트할 수 있게 하는 것이다.

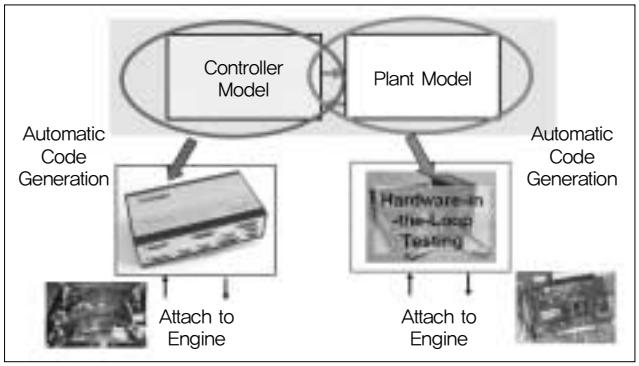
이 RCP와 관련해 실제로 제공되는 제품들이 있는데, 이는 상용 제품과 오픈소스 제품군으로 구분된다. 먼저 상용 제품군에는 MathWorks사의 xPC Target, dSpace사의 AutoBox, National Instruments(NI)사의 LabView등이 있다. RCP 환경은 그 특성상 모델링 툴과 밀접한 연관을 가지고 있는데, NI사의 제품을 제외한 대부분의 RCP 상용 제품은 Simulink로 모델링하며 RTW로 생성된 코드를 자동으로 각 하드웨어에 맞는 디바이스나 실시간 운영체제와 인터페이싱하도록 구성된다. 오픈소스에는 Scilab/Scicos와 Linux RTAI를 활용한 방법 등이 있으나, 전체가 패키지 형태로 묶여 있는 것은 아직까지 등장하지 않은 상황이다.

Hardware-In-the-Loop

Hardware-In-the-Loop(HIL)는 간단히 말해 실제 플랜트 대신에 수학적 모델링을 통해 나온 모델을 실시간으로 실행시켜 외부 인터페이스를 붙이고, 이를 통해 플랜트 이외의 부분에 실제 하드웨어가 있는 것처럼 여기게 하는 것이다. 즉 플랜트의 동역학 부분을 소프트웨어로 대체하는 것이 HIL의 개념이다.

HILS에서도 결국엔 RCP 장비에 임베딩시켜 실행되는 것이므로 앞에서 설명한 RCP 장비를 HILS(Hardware-In-the-Loop System)에서 사용할 수도 있다. 단지 자동 코드 생성되는 모델이 제어기인지 아니면 플랜트인지에 따라서 다른 것이다. 예전에는 제어기도 하드웨어로 구성했으므로, RCP는 HILS에 포함되는 개념으로 생각해도 좋다.

〈그림 2〉를 보면 RCP는 제어기 모델을 자동 코드 생성을 통해 하드웨어에 임베딩함으로써 실제 플랜트를 실시간으로 제어하는 경우이고, HIL은 수학적으로 모델링된 플랜트를 실시간으로 실행해 제어를 위한 로직 등은 실제 하드웨어에 임베딩된 제어기로 제어하는 것으로 실제 하드웨어 플랜트 없이 하는 경우이다.



〈그림 2〉 RCP와 HIL 시스템 설명

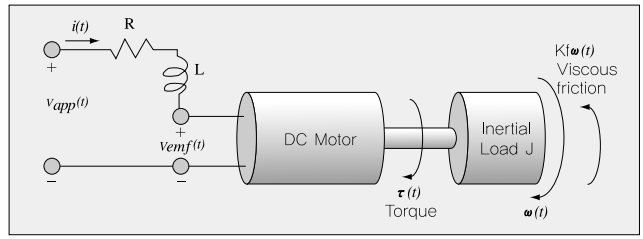
MBD를 활용한 실제 하드웨어 제어

이제 실제 소프트웨어에서 RCP로 제어하는 것을 예로 들어 보자. 여기서 예로 들 예제에서는 UAV(Unmanned Aerial Vehicle)의 전체 모델 중에 Carnard라고 하는 부분을 모델링해 제어기를 설계하고 시뮬레이션을 통해 그 성능을 검증하며 RCP 환경의 하나인 xPC Target을 이용해 제어기의 성능을 확인할 것이다.

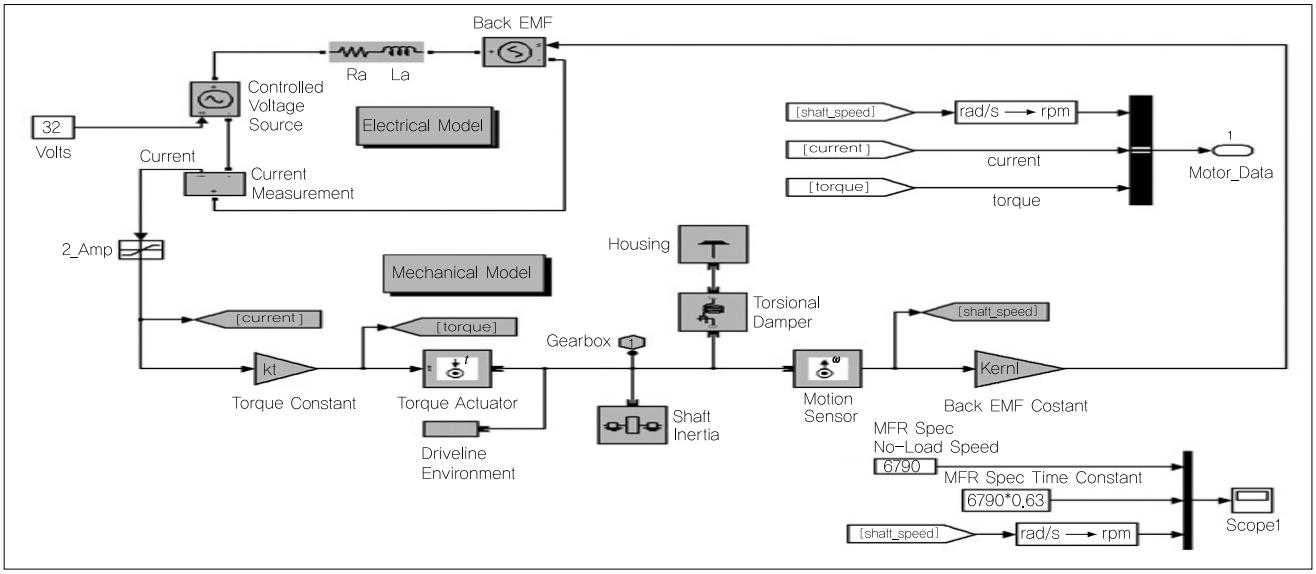
Carnard는 DC 모터를 이용해 제어되고 있으므로, 전체 시스템을 모델링할 때 DC 모터에 대한 부분이 필요하며, DC 모터에 가해지는 힘에 대기 중의 저항이 큰 영향을 미치므로 UAV의 특성상 대기에 대한 모델링도 필요하다. 이러한 것들을 전부 Simulink와 그 외 다른 툴을 이용해 모델링한다. 플랜트와 제어기를 전부 모델링한 후에 시뮬레이션으로 제어기의 게인 값들을 선정하고 xPC Target을 이용해 시뮬레이션 했을 때의 결과와 실시간으로 하드웨어를 제어했을 때의 결과를 비교해 보도록 하겠다.

DC 모터 모델링

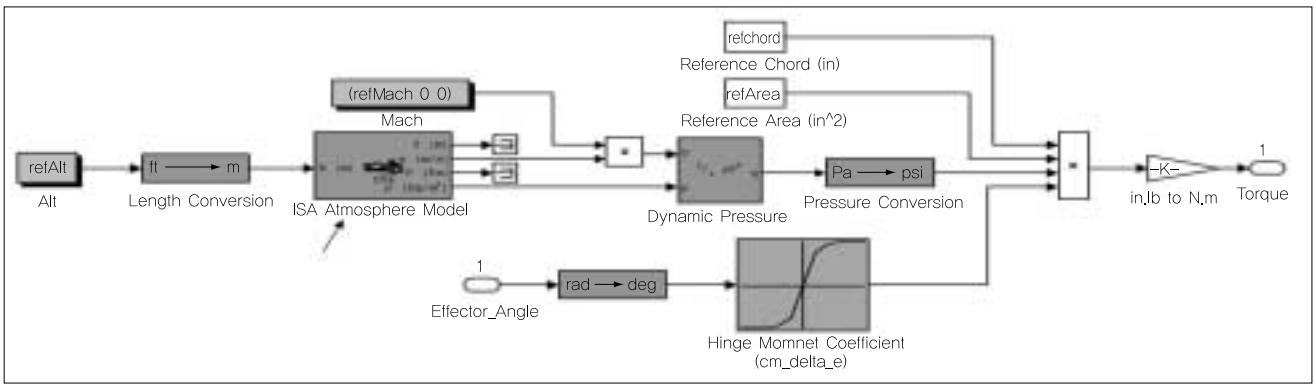
DC 모터는 〈그림 3〉과 같이 단순화할 수 있으며, 이를 수학적 모델링을 하지 않고 Simulink의 add-on 제품 중에 플랜트 모델링을 쉽게 할 수 있게 돕는 Physical Modeling Tool을 이용해 모델링하면 〈그림 4〉와 같다. Aerodynamic Load는 DC 모터에 저항처럼 작용하지만 UAV의 고도에 따라 달라지는 점이 있다. 이런 대기의 모델을 미리 정해 놓은 것이 있어서 이 모델링에서는



〈그림 3〉 DC 모터 다이어그램



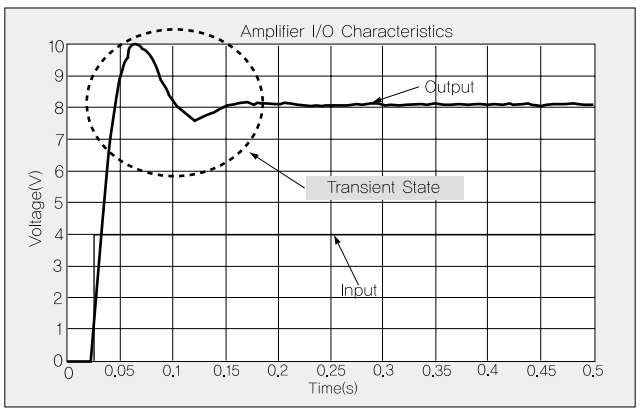
〈그림 4〉 Physical Modeling Tool을 이용한 DC 모터 모델



〈그림 5〉 Aerodynamic Load의 모델링

ISA Atmosphere 모델을 이용했다. 이 부분을 C로 직접 작성할 수도 있지만, 역시나 Simulink의 add-on 중에 하나를 이용하면 〈그림 5〉와 같이 간단하게 하나의 블럭으로 표시할 수 있다.

증폭기 모델링



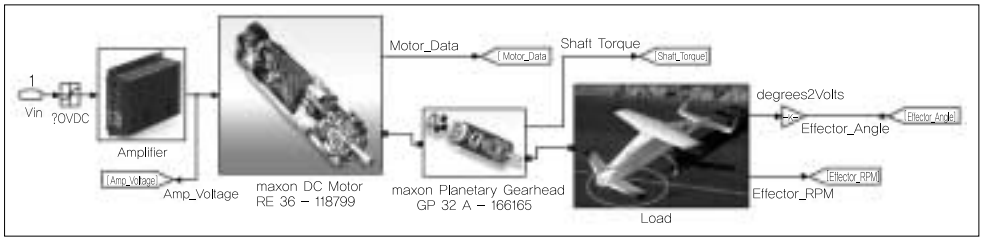
〈그림 6〉 증폭기 측정 데이터

실제로 하드웨어를 구성하게 되면 모터를 구동할 때 증폭기를 사용하게 된다. 증폭기의 특성을 직접 실험으로 테스트해 보면 〈그림 6〉과 같이 steady state일 때는 원하는 만큼 증폭되지만, transient state일 때는 오버슛도 나타낼 수 있으므로 안정화되기 전에는 시스템에 원치 않는 영향을 미치게 된다. 따라서 시뮬레이션 할 때도 이 부분이 고려되어야 한다.

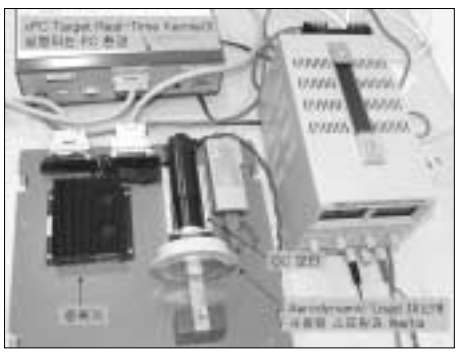
증폭기를 수학적 모델링하는 것은 힘들기 때문에 이미 테스트한 입력 값과 측정된 출력 값을 이용해 모델링한다. System Identification Toolbox를 이용하면 이런 과정을 좀 더 단순하고 간단하게 할 수 있다.

Open-loop System

이런 모델링을 다 모아서 Simulink 모델로 합치고, 같은 역할을 하는 부분끼리 서브시스템으로 묶으면 〈그림 7〉과 같이 나타난다. 이런 Open-loop 모델이 제어하고자 하는 플랜트가 되는 것이다. 이와 동등한 하드웨어 사진을 보면 〈그림 8〉과 같다.



〈그림 7〉 DC 모터, 증폭기, 모터에 걸리는 힘에 대한 Open-loop 모델



〈그림 8〉 DC 모터 하드웨어 구성

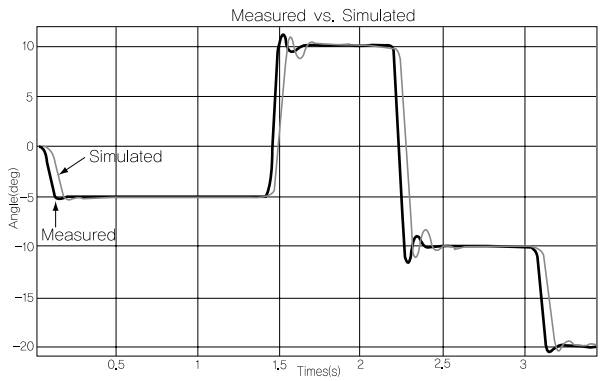
제어기 설계 및 RCP

이제 실제로 알고리즘 개발자가 개발해야 하는 제어기를 설계해 본다. 당연히 Closed-loop로 해야 하므로, Closed-loop 시스템으로 만들고 제어기를 개발할 때는 지난 시간에 설명한 툴들을 이용하게 된다. Simulation으로 원하는 각도로 입력 값을 줬을 때 그와 같은 출력 값이 나오도록 제어기가 설계되었으면, 이제 실시간으로 테스트해야 한다.

실시간 테스트는 xPC Target이라는 툴을 이용해 이뤄지는데, 이는 일반적인 PC에서 운영될 수 있는 xPC Target Real-Time Kernel이라는 실시간 운영체제를 제공한다. 실시간으로 외부 하드웨어와 인터페이싱해야 하므로 A/D 보드 등이 필요한데, xPC Target에서 지정한 보드를 이용하면 이 보드에 대한 모든 소스 코드를 제공하므로 사용자가 이런 디바이스 드라이버 작성에 대해 고민할 필요가 없다. xPC Target이 다른 RCP 툴과 다른 점이라면 일반 PC를 사용하므로 만약 xPC Target에서 지정한 보드 이외의 보드를 써야할 경우 드라이버를 직접 작성해서 계속 재사용할 수 있다는 점이다. 즉 회사에서 직접 만든 보드의 경우 한 번은 드라이버를 작성해야 하지만, 그 뒤로 그 보드를 쓸 때는 계속해서 재 사용할 수 있는 것이다. 다른 RCP를 지원하는 소프트웨어는 이런 개방적인 형태로 되어 있지 않은 게 대부분이다. 즉, 자신들만의 하드웨어를 써야 하는 것이다.

Simulink로 만든 모델을 RTW를 이용해 코드를 생성하고, Microsoft Visual C++ 컴파일러를 이용해 실행 파일을 만들어서 Target PC에서 실행할 수 있게 다운로드한다. 즉 모델을 만드는 호스트 PC(윈도우XP 이상)가 있으며 실시간으로 실행되는 타깃 PC(xPC Target Real-Time Kernel)가 있어야 한다. 두 PC는 서로 TCP/IP로 연결된다. 타깃 PC의 조종은 전부 호스트 PC에서 Simulink와 MATLAB 환경에서 하게 되며 타깃 PC를 조정하기 위한 xpcexplr이라는 GUI 환경을 제공한다.

〈그림 9〉를 보면 시뮬레이션 결과와 실제 하드웨어 테스트 결과가 거의 유사함을 알 수 있다. 이와 같이 MBD를 이용해 최대한의 시뮬레이션을 일반적인 PC상에서 수행하고 어느 정도 만족할 만한 결과가 나왔을 때 RCP로 실시간에서의 성능을 테스트하며 실제로 보드와 같은 환경에 임베딩할 때는 RTW-EC로 C 코드를 생성해 사용하면 된다.



〈그림 9〉 시뮬레이션 결과 및 RCP 결과

지금까지 총 4회에 걸쳐 MBD에 대한 내용을 소개했다. MBD를 제품 개발에 도입하면 시간과 비용의 절감을 가져 올 수 있고 여러 가지 툴을 사용해 최대한 편리한 환경에서 개발할 수 있으므로 한번쯤은 도입을 고려해 보길 바란다. 각 회에서 설명한 툴 등은 얘기한 것보다 훨씬 더 많은 기능들을 가지고 있으므로 그에 대한 궁금한 점이 있다면 각각의 벤더나 필자에게 연락하길 바란다. MBD를 이용한 개발은 국내 소프트웨어 개발자들에게는 생소할 수 있지만 구글과 같은 검색 엔진에서 MBD를 검색해 보면 얼마나 많은 곳에서 사용하는지를 금세 알 수 있다. RTW와 RTW-EC를 위한 모델과 소스 코드가 이달의 디스켓으로 제공되니 생성된 코드가 어떤지 궁금하다면 그 소스 코드를 열어 보길 바란다.

이달의 디스켓 : RTW.zip