技術論文 J. of The Korean Society for Aeronautical and Space Sciences 42(5), 398-405(2014)
DOI:http://dx.doi.org/10.5139/JKSAS.2014.42.5.398

정형검증 도구를 활용한 Fly-By-Wire 헬리콥터 비행제어법칙 자동코드 무결성 확보 방안

안성준*, 조인제*, 강혜진**

Secure methodology of the Autocode integrity for the Helicopter Fly-By-Wire Control Law using formal verification tool

Seong-jun An*, In-je Cho* and Hye-jin Kang**
Korea Aerospace Industries*, APPIA Engineering**

ABSTRACT

Recently the embedded software has been widely applied to the safety-critical systems in aviation and defense industries, therefore, the higher level of reliability, availability and fault tolerance has become a key factor for its implementation into the systems. The integrity of the software can be verified using the static analysis tools. And recent developed static analysis tool can evaluate code integrity through the mathematical analysis method. In this paper we detect the autocode error and violation of coding rules using the formal verification tool, Polyspace®. And the fundamental errors on the flight control law model have been detected and corrected using the formal verification results. As a result of verification process, FBW helicopter control law autocode can ensure code integrity.

초 록

내장형 소프트웨어 기술이 항공 및 방위산업과 같은 안전-필수 시스템에 적용됨에 따라 보다 높은 소프트웨어의 신뢰성이 요구되고 있다. 그 중에서 소프트웨어의 무결성은 주로 정적 분석 도구를 이용해 검증이 이뤄지고 있으며 최근에 개발된 정적 분석 도구는 수학적인 분석 방법을 통해 코드의 무결성을 평가하고 있다. 본 연구에서는 정형 검증 도구인 Polyspace를 이용해 자동코드의 결함을 검출하고, 코딩규칙의 준수 여부를 검증하였다. 검증된 결과를 바탕으로 결함을 가진 제어법칙 모델을 수정하여 코드 생성 이전의 원천적인 결함을 제거 가능함을 확인하였고 FBW 헬리콥터 제어법칙 자동생성코드의 무결성을 확보 할 수 있었다.

Key Words : Safety-critical System(안전-필수 시스템), Auto Generated Code(자동생성 코드), Software Verification(소프트웨어 검증)

Ⅰ. 서 론

소프트웨어 기술의 급격한 발전으로 오늘날 모든 분야에서 소프트웨어가 적용되고 있다. 특히 시스템 제어 등의 목적으로 사용되는 임베디드 소프트웨어의 경우 갈수록 복잡해지고 다양한 기능이 요구됨에 따라 임베디드 소프트웨어 신뢰성에 대한 관심 또한 높아지게 되었다. 항공 및

† Received: September 17, 2013 Accepted: April 16, 2014
* Corresponding author, E-mail : tjdwns6@koreaaero.com
http://journal.ksas.or.kr/
pISSN 125-1348 / eISSN 2287-6871

방위 산업에 적용되는 소프트웨어에서 발생하는 결함은 인명 피해를 초래하기 때문에 신뢰성 평가가 매우 중요하다[1]. 신뢰성을 얻기 위한 방법 중 하나로 정적분석이 수행된다. 이 분석 방식은 코드 내부의 잠재적인 오류가 있는지 확인 할 수 있는 방법이며, 보통 상용 정적분석 도구를 이용해 이루어진다.

정적 분석 도구는 원래 표준 또는 코딩 스타일을 강조하기 위해 쓰여 왔다. 이러한 검사는 의도가 명확하지 않은 표현방식의 사용을 방지하고 소프트웨어의 복잡도를 낮춰 현재 혹은 향후에 발생 가능한 결함을 줄여준다. 하지만 최근에는 코딩규칙의 검사뿐만 아니라 좀 더 효과적이고 강력한 오류검출을 위해 소스코드 내부의 변수간 또는 함수간의 관계를 파악해 결함으로 연계될 수 있는 모순된 코드들을 찾아준다[2].

본 논문에서는 MATLAB/SIMULINK로 설계한 Fly-By-Wire 헬리콥터 제어법칙 모델에서 생성된 코드의 코딩규칙과 무결성을 확보하기 위하여 상용 정적 분석 도구를 이용한 검증 프로세스를 제안하였다. 그리고 제안된 프로세스를 통해 자동생성코드를 검증하여 소스코드에 잠정적인 오류를 파악하고, 발생된 오류를 제거해 자동생성코드의 무결성을 확보하는 방법을 제안하였다.

Ⅱ. 소프트웨어 정적분석 기법

2.1. 정적 분석

정적 분석(Static Analysis)은 프로그램을 컴파일하고 실행하기 전에 소스 코드 수준의 어휘, 구문 분석을 통한 문법 구조를 분석하여 소스코드의 정보를 얻어내는 것을 말한다. 정적 분석은 전형적인 결함 이외에 구제적인 특정 변수 값 대신에 추론 가능한 모든 변수의 값을 사용하는 방식을 사용하기도 한다[3].

정적분석은 주로 분석 도구의 도움을 받아 수행되는데 이 도구는 프로그램 코드를 분석(제어흐름이나 데이터 흐름 분석 등)한다. 정적 분석 도구를 통해 발견되는 전형적인 결함은 아래와 같다.

- 정의되지 않은 값으로 변수참조
- 모듈과 컴포넌트 간에 인터페이스 문제
- 사용되지 않는 변수
- 사용되지 않는 코드(Dead Code)
- 코딩 표준 위반
- 보안 취약성
- 코드의 구문규칙(Syntax)위반

정적 분석을 통한 테스팅은 소스 코드의 전체적인 구조 및 구성 요소들 간의 연관관계, 그리고 호출 관계들을 분석함으로써 이러한 결함을 찾기 용이하며, 좀 더 가용성이 높고 부하가 적은 소프트웨어로 가공하기 위한 정보를 제공한다[4].

FBW 헬리콥터 제어법칙 자동생성 코드를 검증하는 정적분석 도구는 소프트웨어 안전 공학에서 요구하는 코드의 안전성 획득을 위해 정형검증의 분석법인 추상적 해석 기법을 사용하는 도구이다. 이 도구는 소스코드에서 오류의 존재여부를 찾는 동시에 특정 오류가 없음을 증명하는 분석도구이다. 그리고 모델기반 설계에 적합하도록 Simulink로 구성된 모델과 연계되는 기능을 제공한다.

2.2. 정형 기법

정형기법(Formal Method)은 소프트웨어 공학의 해석 기법의 일종으로 오류가 없는 시스템을 설계하여 시스템의 신뢰성을 높이는데 목적이 있다. 요구명세 및 설계명세를 모두 수학적인 문자나 기호로 명세하여 수학적인 증명 방법으로 설계된 시스템의 다양한 특성 및 신뢰성을 검증하고 있다[5]. 정형기법에는 크게 정형 명세(Formal Specification)와 정형 검증(Formal Verification)으로 나눌 수 있다. 정형 명세는 정형 논리(formal logic) 사용되는 기호 등을 이용하여 시스템이 동작할 환경에 대한 시스템 설계 등을 기술하는 것이다. 정형 검증은 정형 논리에서 제공하는 증명 방법 등을 이용하여 정형 명세를 분석하여 무모순성 및 완전성을 검증하거나 설계가 주어진 가정에서 요구사항을 만족하는지를 검증하는 기법이다[6].

본 논문에서는 정형검증(Formal Verification) 중의 추상적 해석(Abstract interpretation)알고리즘을 이용한 정형검증도구를 이용하여 무결성 검증을 수행하였다.

2.2.1 추상적 해석

추상적인 해석(Abstract Interpretation)은 안전이 중요한 비행기 제어 명령, 자율 우주선의랑데부(randezvous)와 도킹 등과 같은 항공우주 시스템에서 지난 10년간 성공적으로 사용하고 있는 해석방법이다. 정적으로 복잡한 메모리 용도와 구조, 매우 큰 C 프로그램에 런타임 오류의 부재를 확인하는 정적 분석에 이용되고 있으며 이 해석법은 이론에서 실행까지 응용 프로그램의 폭넓은 스팩트럼을 갖추고 있다[7]. 정형 기법 중 정형 검증의 가장 대표적인 해석방법은 추상적

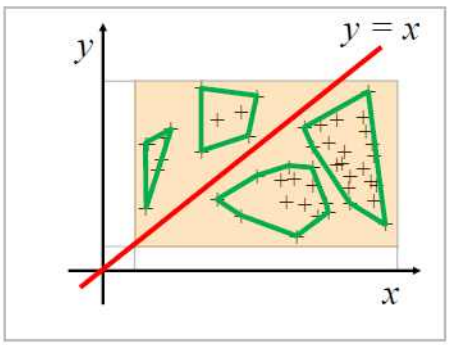

Fig. 1. Abstract interpretation process

해석이며, 이 방법은 런타임 오류가 없음을 증명하는 수학적으로 뛰어난 접근방식이다. 이 해석법은 코드의 각 상태를 분석하는 대신 이러한 상태를 보다 일반적인 형태로 표현 하고 그것을 조작할 수 있는 규칙을 제공해준다. 예를 들어 소스코드 내의 변수의 모든 범위를 확인한 뒤 Division by Zero(ZDV) 오류가 있는지 Fig. 1의 붉은 선과 같이 검사하며 +표시는 변수가 가질 수 있는 모든 가능 범위를 나타낸 것이다. 그 다음 모든 가능 범위에 대하여 if-else, for loop 또는 while loop, switch, inter-procedural operations (function calls)분석에 걸릴 데이터 간의 관계를 표현하기 위해 다면체로 나누며 이것을 기반으로 코드를 분석한다[8].

III.런타임 오류 종류 및 특징

프로그램이 실행되는 도중 오류가 발생하여실행에 치명적인 영향을 주는 것을 런타임 오류라고 한다. 런타임 오류가 발생하면 프로그램의 정지 또는 오동작이 발생함으로 항공기에서는 생명과 연관되는 아주 무서운 오류이므로 정적분석을 통해 런타임 오류가 없음을 증명하는 것이 중요하다. 정적분석을 통해 FBW 헬리콥터 제어법칙 자동생성 코드에서 검사되는 런타임 오류의 종류와 특징은 다음과 같다.

- **Unreachable Code (UNR)**

프로그램이 수행되는 동안 단 한 번도 실행되지 않는 코드이다. 데드코드(Dead Code) 라고도 불리며, 조건문의 논리적 모순에 의해 발생한다.

- **Out of Bounds Array Index (OBAI)**

선언된 배열이 가질 수 있는 인덱스 범위를 벗어나는 경우 발생한다.

- **Division by Zero (ZDV)**

피연산자를 0으로 나누었을 때 발생한다.

- **Non-Initialized Variable (NIV or NLVL)**

선언된 변수를 초기화 하지 않고 사용할 때 발생한다.

- **Overflow (OVFL)**

연산의 결과가 다룰 수 있는 수의 범위를 벗어나는 경우 발생한다.

- **Initialized Return Value (IRV)**

함수의 리턴 되는 값을 이용하여 변수가 초기와 되는 경우가 있는데 만약 함수의 결과 값이 리턴 되지 않은 경우 변수가 초기와 될 수 없어 문제가 발생한다.

- **Shift Operations (SHF)**

정수(int)형 또는 긴 정수보다 얼마만큼 큰 값을 수용 할 수 있는 shift연산이 아닌 경우 발생한다.

- **Illegal Dereferenced Pointer (IDP)**

변수 또는 구조체에 허용되지 않는 포인터 엑세스가 있을 경우 발생한다.

- **Correctness Condition (COR)**

함수포인터가 유효한 함수나 유효한 프로토타입 함수를 가리키고 있지 않다면 발생한다.

- **Non-Initialized Pointer (NIP)**

포인터 변수를 선언하고 초기화 하지 않고 사용할 경우 발생한다.

- **User Assertion (ASRT)**

사용자가 지정한 디버깅함수가 유효하지 않을 경우 발생한다.

- **Non-Termination of Call (NTC)**

프로시저가 무한루프 상태여서 종료될 수 없거나 다른 프로시저를 호출하였을 때 호출된 프로시저가 무한루프 상태인 경우 발생한다.

- **Known Non-Termination of Call (K_NTC)**

NTC와 같은 경우에 발생하며 문제가 발생 한 것을 main에 알려준다.

- **Non-Termination of loop (NTL)**

loop(for, do-while or while)이 종료되지 않았을 때 발생한다.

- **Standard library Function Call (STD_LIB)**

C표준 라이브러리에 유효 하지 않는 함수의 콜을 사용할 경우 발생한다.

- **Absolute Address (ABS_ADDR)**

절대 주소로 접근하기 위해 포인터를 사용할 때 접근이 가능 한 절대주소가 아니라면 문제가 발생한다.

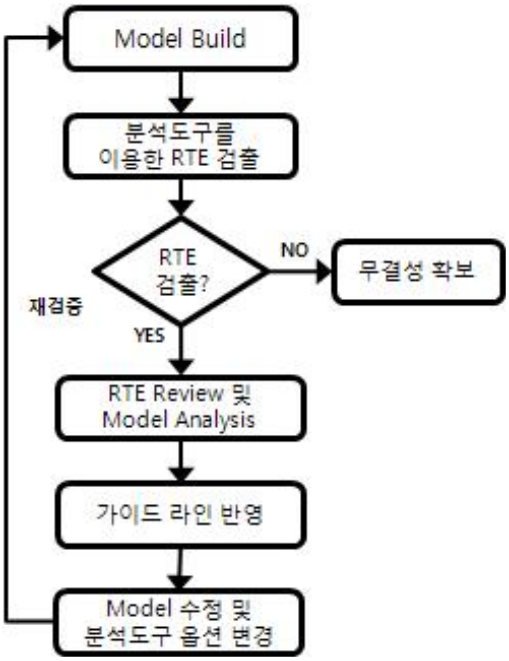

Fig. 2. AutoCode Validation procedures

IV. 자동코드 무결성 검증 절차 및 환경

4.1. 자동코드 무결성 검증 절차

상용 검증도구를 이용하여 FBW 헬리콥터 제어법칙 자동코드의 무결성을 검증하기 위한 절차는 Fig. 2와 같다. 먼저 모델에서 생성된 자동코드를 상용검증도구를 이용하여 런타임 오류가 있는지 분석한다. 런타임 오류가 검출되면 발생한 문제에 대하여 개발자에게 보고하고, 보고된 결과를 바탕으로 검증도구 내부의 문제인지 제어법칙 모델의 설계결함 인지를 파악 하는 분석 과정을 거친다. 분석된 부분은 제어법칙 가이드라인에 반영이 되고 제어법칙 모델의 수정이 이루어진다. 그 결과를 바탕으로 다시 모델검증이 이루어지고, 이 과정을 런타임 오류가 검출되지 않을 때까지 반복적으로 수행하여 자동생성코드의 무결성을 검증한다.

4.2. 자동코드 검증 환경

상용 검증도구는 서버(Server)가 큐 매니저(Queue Manager)를 통해 클라이언트로부터 검증 요청을 받고, 서버는 런타임 체크 후 완료 메시지를 보내준다. 클라이언트는 서버로부터 검증결과를 받아 확인하는데 이 때 자신이 요청한 검증만 다운로드 받을 수 있다. 본 검증에서는 클라이언트 2대, 서버 1대를 이용하여 Fig. 3과 같이 구성하였다. 클라이언트1이 서버로 큐 매니저

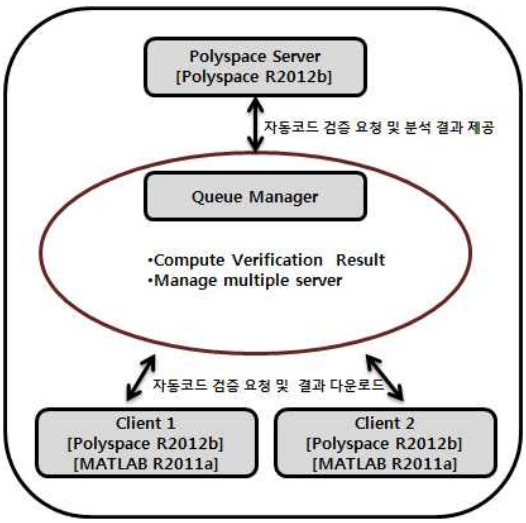

Fig. 3. Helicopter FBW Autocode verification process

를 통해 검증 요청한 것은 클라이언트 1만 결과를 다운받아 확인가능하며 클라이언트 2는 확인해볼 수 없다. 또, 클라이언트1이 자동코드 검증을 서버에 요청해서 검증 수행 중일 때, 클라이언트 2가 서버에 요청할 경우 클라이언트1의 검증이 완료 될 때까지 클라이언트2의 검증요청은 대기 상태가 된다.

V. 자동코드 검증

5.1. 런타임 오류 검증

검증도구를 이용하여 개발 초기에 소스코드에 런타임 오류의 존재여부를 검증하고, 코딩규칙 준수여부를 검사한다. 소스코드를 검사하여 잠재적인 결함이 발생 할 수 있는 부분을 파악하여 각 라인별 코드의 상태를 색상으로 구분하여 준다. 개발자는 검증결과를 바탕으로 오류를 일으킬 가능성이 있는 코드에만 집중할 수 있으므로 검증 시간이 단축될 수 있다.

검증 결과는 녹색(Green Code), 적색(Red Code), 회색(Gray Code), 주황색((Orange Code)으로 색상은 표시되며 녹색은 런타임 오류가 없는 안전한 코드이고 적색은 런타임 오류가 발생하는 코드이다. 회색은 한 번도 실행될 수 없는 데드코드를 의미하며 주황색은 런타임 오류를 발생시킬 가능성이 포함되어있는 코드이다. 녹색을 제외한 색상의 코드에 대하여 어떤 종류의 런타임 오류가 발생하는지에 대하여 보고한다. FBW 헬리콥터 제어법칙 자동 코드에서 몇 가지 회색 코드와 주황색 코드가 발생하였다.

5.1.1 회색(Gray) 코드 검증

소스코드에서 반드시 제거해야하는 적색코드 다음으로 주의해야 할 코드는 회색코드이다. 회색코드는 데드코드로서 논리적으로 실행될 수 없거나 필요 없는 코드이다. 회색코드를 제거 하지 않는다면 향후 프로그램 실행 도중 어떠한 문제가 발생할지 알 수 없으므로 개발자는 회색코드의 근본적인 원인을 파악하여 제거될 수 있도록 해주어야 한다.

```
493   /* Switch: '<S43>/Switch' incorporat
494    *  Gain: '<S43>/Gain'
495    */
496   if (0.0 > 0.0) {
497     vd_tin = -2.5 * local_Sum;
498   } else {
499     vd_tin = local_ph;
500   }
```

Fig. 4. Unreachable Code (Gray Code)

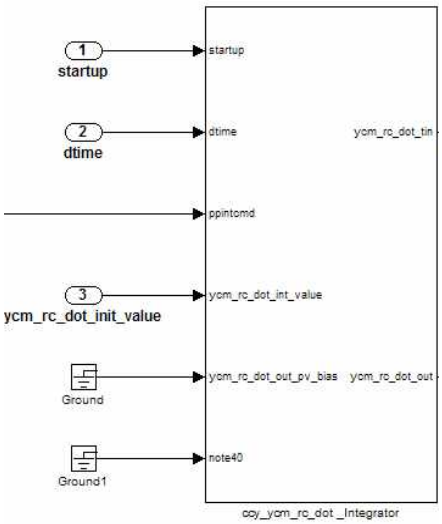

Fig. 5. Unreachable Model

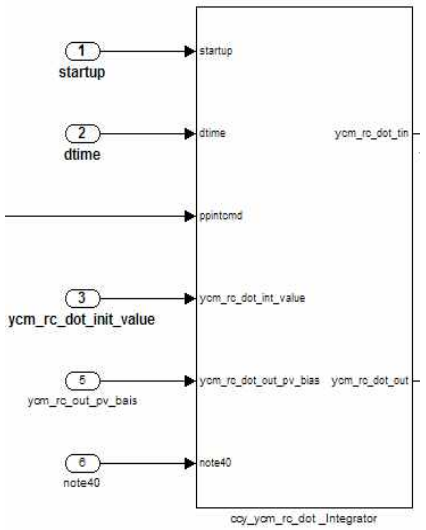

Fig. 6. Modified Unreachable Model

FBW 헬리콥터 제어법칙 자동 코드에서 회색코드는 Fig. 4와 같이 발생되었다. 496라인에 if문에서 스위치 조건에 따라서 로컬변수에 게인값을 곱해주는 부분에 대해서 만족될 수 없는 조건문을 자동코드를 생성하였다. 이 문제는 Fig. 5와 같이 모델에 사용되는 스위치블록의 입력 값이 Ground 블록을 사용함으로써 회색코드가 발생하게 되었다. 이 문제를 해결하기 위해 Fig. 6과 같이 Ground 블록을 제거하고 최상위 블록까지 입력포트를 연결하여 회색코드를 제거 하였다.

이와 같은 문제는 모델 가이드라인을 따라서 모델이 개발되었다면 자동생성코드 검증단계까지 진행되기 이전에 수정될 수 있는 문제로써 향후 제어법칙 모델 개발 시 설계단계에서 다양한 방법으로 모델검증을 수행할 예정이다.

5.1.2 주황색(Orange) 코드 검증

녹색과 적색, 회색으로 표시된 코드들은 확실히 잠재적인 오류의 유무에 대해 확인이 되지만 주황색으로 표시된 코드는 잠재적인 오류발생의 여부가 검증되지 않은 코드이다. 주황색 코드가 발생할 경우 개발자가 발생 원인을 분석하여 런타임 오류를 발생시킬 수 있는 확률에 대하여 판단해야 한다. 만약 잠재적인 오류를 발생시킬 확률이 있다면 Comment처리를 하거나 해당 문제를 제거하여 잠재적인 문제를 해결 할 수 있도록 해야 한다.

5.1.2.1 사용자 개발 자동코드

본 과제에서 주로 발생한 주황색 코드는 오버플로우(Overflow) 결함으로서 변수에 대한 실행 가능범위가 정의되지 않아서 발생하였다. 제어법칙 입력으로 사용되는 각 변수가 가질 수 있는 모든 범위로 설정되었기 때문에 최대값/최소값 연산중 오버플로우 결함이 발생하여 주황색 코드로 분류되었다. Fig. 7과 같이 (-)연산 우측의 제어법칙 입력변수에서 발생하였다. 검증도구에서 지원하는 Data Range Specifications (DRS)라는 기능을 자동 검증 도구의 옵션에 추가하여 오버플로우를 발생시키는 변수의 실행 범위를 지정해

```
690   /* Sum: '<S118>/Sum1' incorporates
691    *  Inport: '<Root>/imu_y'
692    */
693   local_y_err = local_y_k - imu_y;
```

Fig. 7. Overflow Code (Orange Code)

```
imu_u -10000 10000 init
imu_v -10000 10000 init
imu_h -10000 10000 init
imu_y -10000000 10000000 init
imu_x -10000000 10000000 init
imu_r -10000 10000 init
imu_p -10000 10000 init
imu_w -10000 10000 init
imu_q -10000 10000 init
imu_phi -10000 10000 permanent
imu_theta -10000 10000 permanent
imu_psi -10000 10000 permanent
```

Fig. 8. Data Range Specification

줌으로써 검증도구에 추가적인 정보를 제공하여 런타임 오류가 발생하지 않음을 증명할 수 있었다.

Figure 8과 같이 이 기능을 통해 Simulink 모델의 입력 변수의 실행범위를 설정하여 오버플로우 결함으로 인한 주황색 코드를 제거 할 수 있었다. DRS를 사용하는 방법은 원천적인 문제를 해결하는 방법은 아니다. 검증 도구가 결함 발생 여부를 증명하는데 필요한 정보를 제공하는 방법으로 모델에서 근본적인 문제 해결 방안이 필요하다. 이러한 입력에 대한 간단한 오버플로우 결함은 제어법칙 내부의 리미터로 예방할 수 있는 문제이고 좀 더 원천적인 문제를 해결할 수 있는 방법으로 판단된다. 좀 더 복잡한 로직 수행 시 발생할 수 있는 오버플로우 결함에 대한 연구가 필요하다.

5.1.2.2 시뮬링크 라이브러리 자동코드

사용자가 개발한 모델에서 발생하는 결함이 아닌 Simulink처럼 개발도구에서 자체적으로 제공되는 라이브러리에서도 Fig. 9와 같은 주황색

```
56    frac = (u0 - bp0[iLeft]) / (bp0[iLeft + 1U]
57  } else {
58    iLeft = maxIndex - 1U;
```

Fig. 9. Overflow Code(Look-up Table)

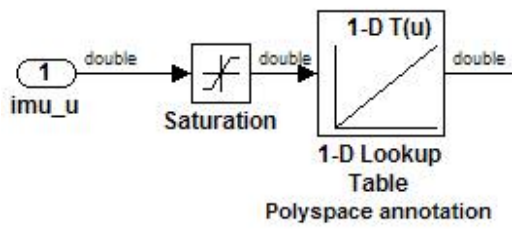

Fig. 10. Annotation(Polyspace)

코드가 발생하였다. Simulink의 Look-up Table의 코드 내부에선 배열 방 하나의 값을 가지고 계산을 하도록 설계되어 있지만 검증 도구에서는 배열 0번방부터 n번방까지의 값을 Look-up table의 범위로 인식하였다. 이 경우 배열 방 하나의 값을 가지지 않고 범위로 인식하여 수행되므로 오버플로우가 발생될 수 있는 문제가 있다. 수학적인 계산에선 오버플로우가 발생할 확률이 있는 오류이지만 해당 코드를 사용자가 검토한 결과 런타임 오류가 발생되지 않는 코드이다.

Figure 10과 같이 FBW 헬리콥터의 제어법칙 모델이 확립될 때 까지 Simulink 모델에 본 과제에 사용된 상용 검증도구가 지원하는 기능인 Annotation을 사용해 Comment처리를 해두는 것으로 오류가 없음을 증명하였다. 이러한 문제는 사용자의 판단이 필요한 부분으로 사용자는 해당 문제의 원인을 파악하고 발생된 문제가 사용자의 코드의 무결성에는 영향이 없음을 확인 하여야 한다.

5.2 자동코드 코딩규칙 검증

FBW 헬리콥터 코딩규칙은 MISRA AC AGC 코딩규칙과 참조항공기 코딩규칙을 적용하여 선정하였다. FBW 헬리콥터의 코딩규칙은 참조항공기 탑재비행제어 소프트웨어 개발에 사용되었던 코딩 규칙을 기본으로 하여 MISRA AC AGC규칙중 off된 규칙을 적용하여 만들어 졌다. MISRA AC AGC코딩규칙은 MISRA C 코딩규칙의 서브셋의 개념으로 자동코드를 위해 MISRA C코딩규칙에서 파생된 코딩규칙으로 Table 1의 13.3과 15.3의 규칙이 MISRA C코딩규칙에선 필요하지만 MISRA AC AGC 코딩규칙에선 불필요하여 off시켰다.

본 과제에서 코딩규칙 준수를 위해 모델의 옵션 중 괄호설정과 관련된 부분이 수정되었으며, 다양한 장비와 인터페이스를 위해서 작성된 Custom Template File에서 모델의 정보를 받아

Table 1. MISRA AC AGC Coding Rule

No	Rule	
13.3	Floating-point expressions shall not be tested for equality or inequality	off
15.3	The final clause of a 'switch' statement shall be the 'default' clause	off

와 외부와 인터페이스 할 수 있도록 생성된 wrapper file역시 코딩규칙을 준수를 위해 함수 선언부분을 수정하였다. 단 C에서 지원하지 않는 Boolean 형태의 변수와 Logical Operator를 사용함으로 인하여 코딩규칙 위반이 발생하였으며 이 문제는 공용함수 블록의 알고리즘을 변경하여 해결할 예정이다.

5.3 자동코드 검증 리뷰

자동생성코드를 검증한 결과는 Report로 제공받을 수 있다. 이 Report에는 검증을 위해 설정된 옵션 및 런타임 오류, 코딩규칙에 대한 결과들이 기재되어 있다. 런타임 오류에 대한 Summary는 색상코드에 대한 각각의 결과를 수량화 하여 몇 퍼센트의 검증이 이루어 졌는지 나타낸다. 런타임 오류에 관한 결과만 Report 파일로 추출 할 수 있고, 코딩규칙에 대한 결과만 Report파일로 추출할 수 있다.

Figure 11과 12에서 나타나듯이 현재 FBW 헬리콥터 제어법칙 자동코드를 검증한 결과 MATLAB에서 제공되는 함수(look1_binlxpw.c, rt_modd.c, rt_roundd.c)의 런타임 오류로 인하여 99.4%의 무결성을 만족 하였고, 코딩규칙에관련해선 위에서 언급한 Boolean과 Logic Operator에 의한 2개의 규칙위반을 제외한 나머지 부분은 만족 하는 것을 확인 할 수 있었다. 위에서 발생한 런타임 오류를 해결하기 위하여 개발회사와 협조하고 있으며, 추가적인 알고리즘 변경으로 해결될 예정이다. 코딩 규칙위반에 대해서는 공용함수 알고리즘을 변경할 예정이다. FBW 헬리콥터 제어법칙 모델의형상이 확정될 때까지 자동코드에 대한 검증을 계속 하여 100%의 코드 무결성을 확보할 예정이다.

Warnings	Errors	Total
2	0	2

Fig. 11. Coding Rule check Result

Proven	Green	Red	Grey	Orange
100.0%	172	0	0	0
100.0%	17	0	0	0
100.0%	13	0	0	0
100.0%	13	0	0	0
100.0%	10	0	0	0
99.6%	761	0	0	3
96.6%	86	0	0	3
99.4%	**1072**	**0**	**0**	**6**

Fig. 12. Run-Time Checks Result

Ⅵ. 결 론

자동생성코드의 최종 목표는 모델에서 구현된 제어법칙을 추가적인 검증 없이 비선형 시뮬레이션환경에서 기능 확인 절차만 거친 후 바로 탑재하는 것이다. 그러기 위해서는 자동생성코드에 대한 무결성을 개발초기에 확보하는 것이 중요하다. 본 논문에서는 Simulink에서 생성된 자동생성코드를 상용 검증 도구인 Polyspace라는 정형기법과 결합된 정적 분석 도구를 사용하여 검증절차와 그 검증절차를 따른 무결성 확보 방안을 제시하였다.

현재 Fly-By-Wire 헬리콥터 제어법칙 자동생성코드 개발에 적용하고 있으며, 결함을 수정한 모델과 코드에 대한 검증이 이루어지고 있으며 모델의 개발 완료까지 런타임 환경에서의 검출 가능한 모든 오류를 자동코드생성 가이드라인과 검증도구를 통해 제거한다면 보다 효율적인 자동생성코드의 무결성을 확보할 수 있을 것이라 판단된다.

후 기

본 연구는 지식경제부가 주관하는 산업융합원천기술개발사업(신산업·주력산업)의 일환으로 수행 되었으며 연구를 지원해주신 관계자 여러분께 감사드립니다.

References

1) Ki-Du Kim, A Study on Reliability Evaluation for Embedded Software, The Institute of Internet, Broadcasting and Communications, Vol.9, No.3, 2009

2) Sung-jin Park, Preparation for dynamic testing and failure detection of Safety-cretical software using a Advanced static analysis tool, Embeddedworld, No.11, 2010

3) Seung-Hwa Song, Graphical Presentation Model for Static Analysis of Software, Korea Computer Congress, Vol.34, No.1, 2007

4) Won-Il Gwun,Practical Software Testing Foundation, STA Consulting Inc., 2010

5) John Rushby, Formal Method and the certification of Critical Systems, Technical Report CSL-93-7, SRI International, Menro Park, CA, 1993

6) Chang-Hun Sung, Development Methodology of Safety-Critical System Using Formal Method, Vol.27. No.2 , 2000

7) P.Cousot, Formal Verification by Abstract interpretation, 4th NASA Formal Methods Symposium(NFM2012), Lecture Notes in Computer Science Volume 7226, 2012, pp 3-7

8) Code Verification and Run-Time Error Detection Through Abstract Interpretation, white paper, mathworks