論文

# 자동 코드생성을 이용한 무인기용 OFP의 검증에 관한 연구

조상욱*, 최기영**

# A Study on Validation of OFP for UAV using Auto Code Generation

Sangook Cho* and Keeyoung Choi**

## ABSTRACT

MATLAB Autocode generation is a feature that converts a block diagram model in Simulink to a c program. Utilizing this function makes MATLAB/Simulink an integrated developing environment, from controller design to implementation. It can reduce development cost and time significantly. However, this automated process requires high reliability on the software, especially the original Simulink block diagram model. And thus, the verification of the codes becomes important. In this study, a UAV flight program which is generated with Simulink is validated and modified according to DO-178B. As a result of applying the procedures, the final program not only satisfied the functional requirement but is also verified with structural point of view with Decision Coverage 93%, Condition Coverage 95% and MC/DC 90%.

## 초 록

자동 코드생성이란 MATLAB의 Simulink 환경에서 설계한 블록 다이어그램을 c 코드로 변환시켜 주는 기능으로 MATLAB과의 연동을 통해 소프트웨어 설계부터 검증까지의 통합된 개발환경을 제공함으로서 개발 시간을 크게 줄일 수 있다. 하지만 생성된 c 프로그램을 무인기에 탑재하기 위해서는 소프트웨어의 신뢰성 확보가 필요하며 특히 원본이 되는 Simulink 블록의 검증이 중요하다. 본 연구에서는 자동 코드생성 기능을 고려하여 Simulink 환경에서 구성한 무인기용 자동비행 프로그램을 DO-178B에 명시된 소프트웨어 시험 과정에 따라 검증하였다. 이 과정을 통해 최종 프로그램은 기능 요구사항에 대한 만족함을 확인했을 뿐 아니라 Decision Coverage 93%, Condition Coverage 95% 그리고 MC/DC 90%로 구조적 측면에서 검증을 수행했다.

**Key Words :** Model-Based Design(모델기반 설계), Code Generation(코드생성), Software Verification(소프트웨어 검증), DO-178B

## Ⅰ. 서 론

MATLAB은 제어기를 설계할 수 있는 환경과 시뮬레이션 환경을 제공함으로써 선박, 자동차, 항공 등 여러 분야에서 많이 사용하는 개발 도구 중 하나이다. 또한 MATLAB Simulink 환경에서

† 2008년 7월 10일 접수 ~ 2009년 3월 24일 심사완료
* 정회원, 인하대학교 항공우주공학과 대학원
** 정회원, 인하대학교 항공우주공학과
교신저자, E-mail : kchoi@inha.ac.kr
인천광역시 남구 용현3동 253번지

설계된 제어기는 자동 코드생성 기능을 통해 c 또는 c++의 코드로 변환이 가능하다. 이는 Simulink 환경에서 구현한 알고리즘을 담당 엔지니어가 직접 다른 언어로 변환하는 기존의 과정에 비해 개발 시간 및 비용을 크게 줄일 수 있으며 이미 다양한 분야에서 사용되고 있다.

코드생성 도구에 대한 신뢰성은 이미 다양한 시스템 개발 사례를 통해 확인할 수 있다. 예를 들어 항공분야의 경우 MATLAB의 자동 코드생성 도구는 록히드 마틴에서 JSF의 비행제어 시스템을 설계하고 개발하는 과정에서 사용되었으며 그 신뢰성을 입증한바 있다. 하지만 코드생성 기능은 단순히 Simulink 블록을 다른 언어로 바꾸어주는 것일 뿐이므로 잘못된 블록을 구성했을 경우 생성된 코드도 반드시 오작동을 일으키게 되어있다. 그러므로 생성된 코드를 시스템에 탑재하여 사용하기 위해서는 무엇보다 원본이 되는 Simulink 블록에 대한 엄격한 검증과정이 필요하다.

아래 그림과 같이 Simulink 환경에서 구현한 알고리즘은 c 코드로 변환하기 전에 시험조건(Test Case)을 정의하여 원본 모델에 대해 충분히 검증을 수행한다. 검증이 완료된 원본 모델은 자동 코드생성을 통해 c 코드로 변환하며 다시 이에 대해 앞에서 정의한 시험 조건들을 바탕으로 원본 블록과의 동일성에 대해 검증한다. 그 이유는 사용자의 Real-Time Workshop 설정에 따라 최적화가 이루어지기 때문에 생성된 c 코드의 구조가 Simulink 블록과 다를 수도 있기 때문이다[2]. 그러므로 생성된 코드의 구조가 변했을 경우 검증 결과를 바탕으로 추가 시험조건을 정의하여 원본 블록과 동일한 수준의 목표 Coverage 결과가 나올 때까지 테스트를 수행한다. 마지막으로 생성된 코드에 대해 추가로 정의한 시험 조건들은 다시 원본 블록에 적용하여 테스트 결과가 일치한지를 확인한다.

본 연구에서는 MATLAB 환경에서 원본 Simulink 블록에 대하여 항공 분야에서 주로 사용되는 소프트웨어 개발 규격서 DO-178B에 근거해 검증절차를 적용하였다.

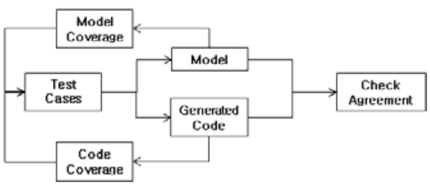


그림 1. 생성된 코드의 검증과정[1]

## II. 본 론

### 2.1 소프트웨어 신뢰성과 규격서

소프트웨어의 신뢰성 확보는 소프트웨어의 품질을 기반으로 하는 것이며 이를 위해서는 규정된 개발 절차를 따르는 것이 중요하다. 즉 소프트웨어 신뢰성의 특성을 파악하고 개발하고자 하는 분야에 적합한 규격 및 절차를 따랐을 때 소프트웨어를 사용하기 위한 최소한의 신뢰성을 확보했다고 할 수 있다. 그러므로 소프트웨어 시험을 수행하기 위해서는 해당 소프트웨어가 사용될 분야 및 환경을 고려하여 테스트 방법을 결정해야 한다. 이러한 환경 또는 목적을 만족하기 위하여 각 분야 별 소프트웨어 규격서가 존재한다. 예를 들어 자동차 산업의 경우 MISRA가 있으며 철도 분야에는 EN 50126이 있다[3]. 본 연구는 방산 및 항공 분야에서 사용되는 RTCA DO-178B[4]를 바탕으로 하였다.

DO-178, Software Considerations in Airborne Systems and Equipment Certification[4]은 RTCA에서 항공기 탑재 부품 및 시스템의 소프트웨어 개발을 위해 만든 규격서로 감항(Airworthiness) 요구조건의 만족을 목표로 하고 있다. 본 연구에서는 검증절차에 대해 서술한 6장 소프트웨어 검증에 대한 내용을 바탕으로 하였다.

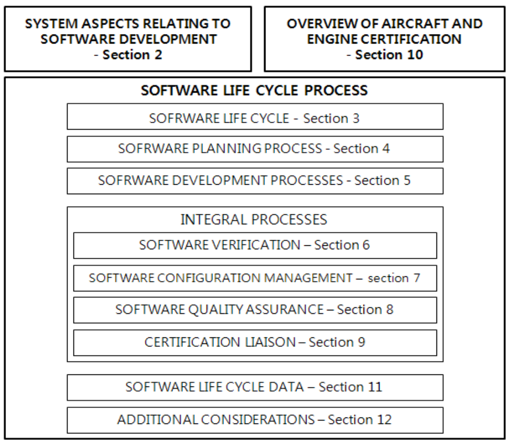


그림 2. DO-178B의 구성[4]

### 2.2 소프트웨어 레벨 정의

소프트웨어 내의 오류는 시스템의 고장 상태를 유발할 수 있으며 이러한 고장 상태의 위험 정도에 따라 소프트웨어의 레벨이 결정된다.

표 1. 소프트웨어 레벨에 따른 Coverage Level

| Software Level | Structural Coverage |
|---|---|
| Level A | MC/DC, DC & SC |
| Level B | DC & SC |
| Level C | SC |
| Level D | Not Required |
| Level E | Not Required |

다음은 DO-178B에서 명시한 소프트웨어의 고장 상태의 종류 및 정의이다.

- Catastrophic Failure Condition

안전한 비행이나 이착륙을 수행할 수 없는 고장 상태로 이것을 일으킬 수 있는 소프트웨어는 Level A로 분류된다.

- Hazardous/Severe-Major Failure Condition

시스템의 고장에 대처하기 위하여 항공기의 능력 또는 승무원의 능력이 상당히 감소되며 잠재적으로는 심각한 손상을 일으킬 수 있는 고장 상태로 이것을 일으킬 수 있는 소프트웨어는 Level B로 분류된다.

- Major Failure Condition

시스템의 고장에 대처하기 위하여 항공기의 능력 또는 승무원의 능력이 상당히 감소되며 안전성 여유 및 기능이 상당히 줄어든 고장 상태로 이것을 일으킬 수 있는 소프트웨어는 Level C로 분류된다.

- Minor Failure Condition

항공기의 안전성은 크게 줄어들지 않지만 이를 대처하기 위하여 주어진 범위 내에서 승무원의 능력수행이 필요한 고장 상태로 이것을 일으킬 수 있는 소프트웨어는 Level D로 분류된다.

- No Effect Failure Condition

항공기의 동작 능력이나 승무원의 업무 부담이 발생하지 않는 고장 상태로 이것을 일으킬 수 있는 소프트웨어는 Level E로 분류된다.

각각의 고장상태의 위험도에 따라 해당 소프트웨어의 레벨이 결정이 되며 각 소프트웨어 레벨에 따라 표 1에서 명시한 종류의 Coverage를 만족해야 한다. 위의 표에서 SC는 Statement Coverage를 DC는 Decision Coverage를 그리고 MC/DC는 Modified Condition/Decision Coverage를 의미한다.

## 2.3 소프트웨어 시험 과정

검증(Verify)은 시험(Test), 분석(Analysis) 그리고 검토(Review)로 이루어져 있다. 분석은 소프트웨어가 옳다는 증거를 제시하는 과정이며 검토는 옳다는 것에 대하여 정성적인 자료를 만드는 것으로 이들은 시험을 하기 전에 수행된다. 이때 검토와 분석이 수행되는 대상은 다음과 같다.

- 상위, 하위 레벨 요구사항
- 소프트웨어 구조
- 소스 코드
- 통합 과정에서의 결과
- 시험조건(Test Case), 절차 및 결과

소프트웨어 시험 과정(Software Testing Process)을 수행하기 위해서는 우선 시험조건(Test Case)을 정의해야 한다. 시험조건은 시험 대상에 대한 입력, 실행 조건 그리고 예상되는 결과로 구성되어 있으며 이는 우선적으로 소프트웨어 요구사항(Software Requirement)을 기반으로 만들어져야 한다. 시험조건을 만드는 목적은 대상의 정확한 작동을 검증하고 잠재적으로 발생할 수 있는 오류를 찾아내기 위함이다.

그림 3은 DO-178B에 명시된 소프트웨어 시험 과정이다. 처음 수행할 시험은 하위 레벨 시험(Low-Level Testing)으로 이 과정에서는 하위 레벨 요구사항(Low-Level Requirements)이 소스 코드로 정확히 구현되었는지를 확인한다. 다음으로는 소프트웨어 통합 시험(Software Integration Testing)을 수행하며 소프트웨어 요구 사항과 구성 요소간의 상관관계를 확인하고 이들 간의 명시된 사항들에 대한 구현을 확인하다. 마지막으로 하드웨어/소프트웨어 통합 시험(Hardware/Software Integration Testing)을 수행하며 소프트웨어가 대상 하드웨어 환경에서 정확히 동작하는가를 확인한다.

각각의 시험 대상에 대하여 수행할 시험 항목 중 첫 번째는 기능 검증을 의미하는 Software Requirement Coverage Analysis로 앞서 정의한 기능 요구사항에 대한 만족여부를 확인한다. 이 때 시험의 입력범위는 정상범위 시험조건

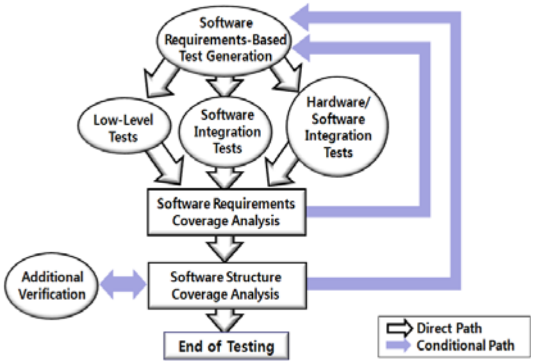


그림 3. 소프트웨어 시험 과정[3]

(Normal Range Test Case)과 강건성 시험조건(Robustness Test Case)을 모두 적용한다.

기능 검증을 마친 소프트웨어는 Software Structure Coverage Analysis를 수행하며 이는 구조적 관점에서의 시험을 뜻한다. 구조적 시험의 목적은 소프트웨어 내에서 수행되지 않는 데드 코드(Dead Code)를 찾아내는 것이며 소프트웨어의 레벨에 따라 지정된 종류의 Coverage를 만족시켜야 한다.

## 2.4 Coverage의 종류 및 특징

소프트웨어 구조적 시험의 최종 목적은 소프트웨어의 구조를 파악하여 내부의 모든 구문이 정상적으로 수행하는지를 확인하고 실행되지 않는 구문을 찾아내 이를 제거하는 것이다. 하지만 시험 결과를 판단하기 위해서는 수치적인 지표가 필요하다. 구조적 시험에 대해 DO-178B에 명시된 Coverage의 종류는 다음과 같다.

1) Statement Coverage

SC(Statement Coverage)는 코드 내에서 각각의 라인이 실행되었는가를 측정하며 Line Coverage 또는 Segment Coverage라 불리기도 한다. 이는 소스코드에 별다른 과정 없이 오브젝트 코드에도 적용시킬 수 있다는 장점을 가지고 있으나 특정한 제어 구조에 대해서는 반응하지 않는다. 이를 100% 만족했다는 것은 프로그램 내의 모든 코드가 실행되었음을 의미한다.

2) Decision Coverage

DC(Decision Coverage)는 논리 구문 내의 논리 연산자에 관계없이 전체 조건문이 참과 거짓을 모두 수행했는지를 측정하며 Branch Coverage라고도 불린다. 이는 Statement Coverage보다 엄격한 시험을 수행할 수 있으며 단순하다는 이점을 가지지만 조건문 내에서 "&&"와 "||" 등의 Short-circuit Operator에 의해 발생하는 분기를 무시한다. 이를 100% 만족했다는 것은 프로그램 내의 각 조건문들에 대하여 참과 거짓인 경우가 모두 실행되었음을 의미한다.

3) MC/DC

MC/DC(Modified Condition/Decision Coverage)는 각 개별조건이 다른 개별조건에 의해 영향을 받지 않고 조건문 결과에 독립적으로 영향을 미치는지를 확인하며 DO-178B Level A의 소프트웨어는 이를 반드시 확인해야 한다. MC/DC의 결정 테이블을 만들기 위한 접근 방법은 아래와 같다.

표 2. Coverage의 예문

```
if (A & B) | (C & D)
    statement1;
else
    statement2;
```

표 3. MC/DC를 만족하기 위한 조건

| Condition | True Out | False Out |
|---|---|---|
| Condition A | TTTF | FTTF |
| Condition B | TTTF | TFTF |
| Condition C | FTTT | FTFT |
| Condition D | FTTT | FTTF |

- 프로그램에 있는 모든 결정 포인트 내의 조건문은 적어도 한번 이상 가능한 모든 결과(참, 거짓)에 대하여 수행되어야 한다.
- 프로그램에 있는 결정 포인트 내의 모든 개별 조건들은 적어도 한번 이상 가능한 결과(참, 거짓)에 대하여 수행되어야 한다.
- 결정 포인트에 있는 각각의 개별조건들은 다른 개별조건에 영향을 받지 않고 그 결정 포인트의 결과에 독립적으로 영향을 주어야 한다.

표 2에서 조건문 내의 개별조건은 A, B, C, D 총 4개로 구성되어 있다. 그러므로 MC/DC를 100% 만족하기 위해서는 최소한 표 3의 조건들에 대하여 시험을 수행해야 한다.

표 3에서 T는 True를 F는 False를 의미한다. Condition A를 보면 B, C 그리고 D의 상태를 각각 T, T, F로 고정시킨 채, A의 상태를 T 그리고 F로 변화시켜 가면서 A가 전체 조건문에 대하여 정상적으로 영향을 미칠 수 있는지에 대하여 확인한다. 이때 표 3의 Condition A에 명시된 조건들은 A의 영향을 확인할 수 있는 경우의 수 중 하나이며, 이 외에도 'TTFF & FTFF', 'TTFT & TFFT' 등의 경우도 A에 대한 시험조건으로 동일하게 사용할 수 있다. MC/DM를 100% 만족했다는 것은 프로그램 내의 모든 개별조건들이 각 조건문에 대해 정상적으로 영향을 미쳤음을 의미한다.

## 2.5 무인기용 OFP 검증

본 연구에서는 Simulink 환경에서 구현한 무인기용 자동비행 알고리즘 블록에 대하여 각 구성요소들의 단위 시험과 소프트웨어 통합 시험을

모두 수행하였으며, 다음은 통합시험 중 고도제어 블록에 대하여 시험한 예이다.

DO-178B에 명시된 시험 과정 중 Requirement Coverage Analysis는 시험 담당자가 기능 요구사항을 바탕으로 검증을 수행하였으며 Structure Coverage Analysis는 Simulink에서 지원하는 Model Coverage Tool[5]을 사용하여 Decision Coverage, MC/DC를 산출하였다. 다음 그림은 검증을 위해 사용한 Model Coverage의 설정을 나타낸 것이다.

본 연구에서는 DO-178B에서 만족하도록 명시된 Decision Coverage와 MC/DC와 함께 추가 자료로 Condition Coverage를 측정하도록 설정하였다. Condition Coverage란 조건문 내의 각각 개별 조건에 대하여 참과 거짓이 모두 수행되었는지를 측정하는 지표이다. 또한 위의 설정을 통해 시뮬레이션을 수행한 결과는 HTML 형식의 보고서로 확인할 수 있다.

비록 대상 프로그램에서 정의한 소프트웨어 레벨에 대하여 DO-178B에서는 Statement Coverage를 만족해야 한다고 명시되어 있지만 Simulink의 특성상 대상 소프트웨어는 코드가 아닌 블록과 신호 선들로 구성되어 있기 때문에 Statement Coverage가 존재하지 않는다. 그러므로 Simulink 환경에서의 검증에서는 Decision Coverage와 MC/DC 만을 확인하였으며

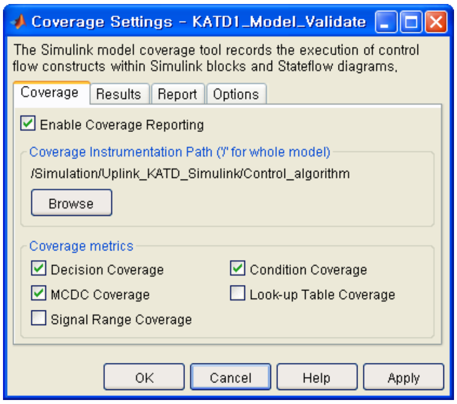


그림 4. Model Coverage 설정[7]

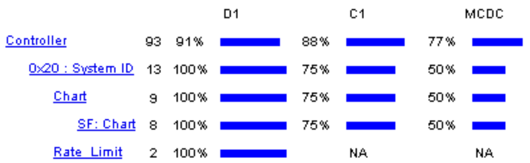


그림 5. 시험조건에 대한 Model Coverage 결과

Statement Coverage는 위의 Coverage 결과들을 바탕으로 각 블록의 실행여부를 통해서 예측하였다.

다음은 시험 과정을 정리한 것이다[6].

① 대상 프로그램에 대하여 분석(Analysis) 및 검토(Review)를 수행한다.
② 기능 요구사항을 바탕으로 시험조건(Test Case)을 정의하여 Requirement Coverage Analysis를 수행한다.
③ 프로그램을 실행하는 동안 내부의 수행 경로를 분석한 후 이를 통해 Decision Coverage와 MC/DC를 산출한다.
④ 산출된 Coverage를 바탕으로 이를 증가시킬 수 있는 시험조건을 추가로 정의한다.
⑤ 추가된 시험 조건들을 수행하여 ③과 동일한 방법으로 Coverage를 산출한다.
⑥ 목표한 Coverage를 만족할 때까지 ④와 ⑤의 과정을 반복한다.

무인기의 자동비행 알고리즘의 경우 소프트웨어의 오작동이 사고를 의미하므로 본 연구에서는 대상 소프트웨어의 레벨을 A로 정의하였으며 DO-178B에 명시된 DC, MC/DC가 90% 이상 만족하는 것을 목표로 하였다.

1) 기능 요구사항 검증

기능 요구사항 검증에서는 정의된 기능 요구사항 항목들을 바탕으로 명시된 기능의 수행여부를 확인하기 위해 총 21개의 시험조건을 정의하였으며 표 4는 전체 블록 중 고도제어 블록의 검증을 위해 사용한 시험조건 중 하나이다.

아래 그림은 위의 시험조건을 바탕으로 시뮬레이션을 수행한 항공기의 고도와 방위각을 나타낸 것이다.

그림 6을 보면 300m의 고도명령에 대하여 항공기가 약 8m의 오차를 가지고 있으며 이때 항공기의 방위각은 명령 값인 180°에서 최대 3°의 오차를 가지고 추종하고 있어 프로그램 개발 시 자체적으로 정의한 기능 요구사항의 만족함을 확인하였다.

2) Structure Coverage 검증

표 4에 대해 시뮬레이션을 수행한 Model Coverage 결과, 고도제어 블록에 대하여 DC가 50%(3/6)로 측정되었다. 이는 전체 블록 중에 고도제어 블록에 대한 Decision의 수는 총 6개이며 이 중 위의 설정으로 시뮬레이션을 수행하는 동안 3개의 경우만 수행되었음을 의미한다.

표 4. 시험조건 1의 시뮬레이션 설정

| |
|---|
| 비행명령 : 고도제어 모드 |
| 고도명령 : 300m |
| 초기 방위각 : 180° |
| 초기 고도 : 213m |
| 기타 항공기의 자세 및 각속도는 0°으로 설정 |

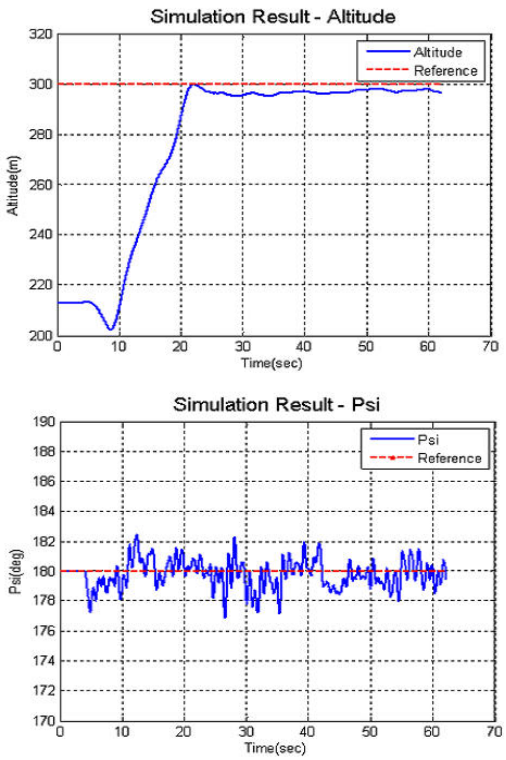


그림 6. 시험조건 1의 시뮬레이션 결과

| Metric | Coverage (this object) | Coverage (inc. descendants) |
|---|---|---|
| Cyclomatic Complexity | 0 | 3 |
| Decision (D1) | NA | 50% (3/6) decision outcomes |

그림 7. 고도제어 블록의 Model Coverage 결과

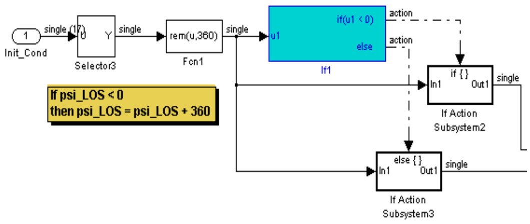


그림 8. 고도제어 블록의 하위블록 : If block "If1"

그림 8은 고도제어 블록에 포함된 조건문 중 하나로 방위각 명령과 현재 항공기의 방위각의 차이 값을 -180°~180°의 범위로 만들어 주는 부분이다. 그림 9에서 볼 수 있듯이 시뮬레이션 동안 if 블록은 거짓의 경우만 수행되어 이 조건문에 대한 Decision Coverage 결과는 50%로 계산되었다.

| Metric | Coverage |
|---|---|
| Cyclomatic Complexity | 1 |
| Decision (D1) | 50% (1/2) decision outcomes |

**Decisions analyzed:**

| | |
|---|---|
| input logical value | 50% |
| false | 2899/2899 |
| true | 0/2899 |

그림 9. If block "If1"의 Decision Coverage 결과

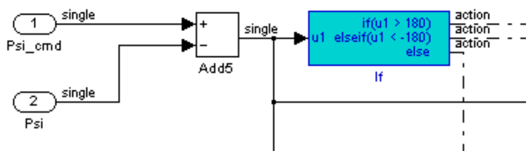


그림 10. 고도제어 블록의 하위블록 : If block "If"

| Metric | Coverage |
|---|---|
| Cyclomatic Complexity | 2 |
| Decision (D1) | 50% (2/4) decision outcomes |

**Decisions analyzed:**

| | |
|---|---|
| input 1 "if" condition | 50% |
| false | 2899/2899 |
| true | 0/2899 |
| input 2 "elseif" condition | 50% |
| false | 2899/2899 |
| true | 0/2899 |

그림 11. If block "If"의 Decision Coverage 결과

그림 10은 고도제어 블록에 포함된 다른 조건문으로 if와 elseif로 구성되어 있다. 이들 if와 elseif에 대하여 각각 참과 거짓에 대해 총 4가지의 Decision이 존재하며 이에 대한 Model Coverage 결과는 그림 11과 같다. 이를 보면 시뮬레이션을 수행하면서 if와 elseif 모두 참인 경우만이 수행되어 이 조건문에 대한 Decision Coverage는 50%로 계산되었다.

그러므로 고도제어 블록에 대한 Coverage를 높이기 위해서는 Model Coverage 결과를 바탕으로 이전 시뮬레이션에서 수행하지 않은 조건문들이 참과 거짓인 경우가 모두 수행될 수 있도록 추가적인 시험조건을 정의해야 한다. 다음 표는 이러한 정보를 바탕으로 정의한 추가 시험 조건들이다.

표 5. 시험조건 2의 시뮬레이션 설정

| |
|---|
| 비행명령 : 고도제어 모드 |
| 고도명령 : 300m |
| 초기 방위각 : 2° |
| 초기 고도 : 213m |
| 기타 항공기의 자세 및 각속도는 0°으로 설정 |

표 6. 시험조건 3의 시뮬레이션 설정

| |
|---|
| 비행명령 : 고도제어 모드<br>고도명령 : 300m<br>초기 방위각 : 359˚<br>초기 고도 : 213m<br>기타 항공기의 자세 및 각속도는 0˚으로 설정 |

| Metric | Coverage |
|---|---|
| Cyclomatic Complexity | 2 |
| Decision (D1) | 75% (3/4) decision outcomes |

Decisions analyzed:

| | |
|---|---|
| input 1 "if" condition | 50% |
| false | 1797/1797 |
| true | 0/1797 |
| input 2 "elseif" condition | 100% |
| false | 1738/1797 |
| true | 59/1797 |

그림 12. 시험조건 2에 대한 If block "If"의 Decision Coverage 결과

시험조건 2의 경우 그림 10에서 Psi_cmd의 값이 0°으로 설정되어 있을 때, 초기에 if 블록으로 들어가는 입력이 2°이므로 그림 11의 Input 2 "elseif“ condition이 참이 된다. 그림 12는 이에 대한 Coverage 결과이다. 이를 보면 추가로 정의한 시험조건이 예상한 경로를 수행하여 Decision Coverage가 증가했음을 볼 수 있다.

동일한 방법으로 시험조건 3을 정의하였으며 이는 그림 12에서 수행되지 못한 경로를 수행하기 위한 조건이다.

그림 10의 Psi_cmd의 값이 0°으로 설정되었을 때 if 블록으로 들어가는 입력이 -359°이므로 그림 12의 Input 1 "if“ condition이 참이 된다. 다음 그림은 시험조건 3을 추가했을 때의 Coverage 결과이다.

| Metric | Coverage |
|---|---|
| Cyclomatic Complexity | 2 |
| Decision (D1) | 100% (4/4) decision outcomes |

Decisions analyzed:

| | |
|---|---|
| input 1 "if" condition | 100% |
| false | 2477/2695 |
| true | 218/2695 |
| input 2 "elseif" condition | 100% |
| false | 2418/2477 |
| true | 59/2477 |

그림 13. 시험조건 3에 대한 If block "If"의 Decision Coverage 결과

하지만 고도제어 블록의 검증을 수행하는 과정에서 그림 8의 조건문 블록은 어떠한 경우라도 참을 수행하는 조건이 존재하지 않았으며 원인분석 결과 고도제어 블록 이전에 신호처리를 하는 부분에서 이와 동일한 기능이 이미 있었기 때문으로 밝혀졌다. 그러므로 이 조건문 블록은 데드코드 (Dead Code)로 판단되어 제거하였다.

고도제어 블록에 대한 검증 과정을 전체 블록에 대해 수행한 결과 Coverage를 90% 이상 만족시킬 수 있는 최종 시험조건은 총 41개로 이들에 대해 시뮬레이션을 수행했을 때 DC 93%, MC/DC 90%의 결과를 얻었다. 이는 기능 요구사항에 대해서만 검증을 수행했던 이전 검증방법의 Coverage 결과인 DC 80%, MC/DC 50%보다 높은 수치로 대상 프로그램의 분기 및 내부 코드들이 더 많은 영역에 대하여 정상적으로 수행되었음을 의미한다. 위의 결과에서 검증되지 않은 분기들은 ‘Rate limit’, ’Saturate’ 블록 내의 조건문 등으로 위와 같은 통합시험을 이용하여 이를 만족하기는 어렵기 때문에 단위 시험을 통하여 추가로 확인하였다.

## Ⅲ. 결 론

본 연구에서는 MATLAB의 Simulink 환경에서 무인기용 자동비행 프로그램을 구성하고 이를 DO-178B에 명시된 내용을 바탕으로 검증을 수행하였다. 이 때 기능 요구사항에 대한 검증뿐만 아니라 구조에 대한 검증을 수행하였으며 각각의 시험조건에 대하여 정의한 요구사항들을 만족함을 확인하였다.

이를 통해 기존의 테스터에 의지한 시험 과정과는 달리 체계화된 시험 방법을 정립할 수 있었으며 시험 결과 Decision Coverage 93%, Condition Coverage 95%, MC/DC 90%로 측정되었다. 또한 만족하지 못한 Coverage들은 추가로 단위 시험을 통해 검증하였다. 이러한 절차를 바탕으로 기능 검증만을 확인했던 이전 검증방법보다 더 높은 신뢰성을 갖는 블록을 구성할 수 있었다.

위의 검증과정을 수행한 블록은 그 신뢰성이 확보되었기 때문에 자동 코드생성을 통해 c 코드로 변환할 수 있다. 추후로는 그림 1에 명시한 절차와 같이 변환된 코드에 대해서도 검증을 수행할 예정이다.

## 후 기

이 논문은 인하대학교의 지원으로 연구된 결과입니다.

## 참고문헌

1) William Aldrich, "Using Model Coverage Analysis to Improve the Controls Development Process", AIAA Modeling and Simulation Technologies Conference and Exhibit, 2002. 8.

2) The Mathworks, "Real-Time Workshop User's Guide", The Mathworks Inc., 2005.

3) 장수주, “소프트웨어 신뢰도 및 테스팅”, 소프트웨어 분석 및 테스팅 교육, 2007. 10.

4) RTCA Inc., "Software Considerations Airborne Systems and Equipment Certification", Document RTCA/DO-178B, December 1992.

5) The Mathworks, "Simulink Verification and Validation 2 User's Guide", The Mathworks Inc., 2007.

6) Andre. C. Coulter, "Graybox Software Testing Methodology Embedded Software Testing Technique", IEEE Digital Avionics Systems Conference, 1999. 10.

7) 김영일 외 3인, “항공전자시스템컴퓨터 탑재소프트웨어 개발“, 한국항공우주학회지, 제 33권 9호, 2005, pp. 104~112.

8) http://www.mathworks.co.kr