한국철도학회논문집 제15권 제2호 ▌pp.122-128 (2012년 4월)
JOURNAL OF THE KOREAN SOCIETY FOR RAILWAY VOL.15, NO.2 ▌pp.122-128 (April 2012)

검수고에서 소프트웨어 결함허용기법을 고려한 가상궤도회로의 적용에 대한 연구

A Study on the Application of Virtual Track Circuit by Considering Software Fault Tolerance Techniques in Depot

이명철[1] · 고영환[2] · 김민석[3†] · 이종우[4]

Myoung-Chol Lee · Young-Hwan Ko · Min-Seok Kim · Jong-Woo Lee

Abstract Considering structure of depot, it is impossible to install the track circuit systems due to iron-beam. Because rails and earth are connected by the iron-beam, there is much leakage current. So, it is hard to apply the track circuit systems. Thus, when trains go to the depot, sign which indicates existence of trains is used manually. In case of wrong sign, accidents occur such as train crash, derailment etc. Currently, location of trains has been found by using optical sensor in the depot to prevent the accidents. However, it costs a great deal to install and maintain the optical sensor. Therefore, this method is hardly used in train operation institutes. In this paper, virtual track circuit systems are introduced by using software program in the depot. Also, algorithm of the virtual track circuit systems is proposed. In case that signal is handled to the depot which is occupied by the train, safety is ensured by indicating sign which means existence of trains and stop signal. Also, proper fault tolerance techniques are proposed to the software by analyzing reliability and availability.

Keywords : Virtual track circuit, Railroad signaling system, Software fault tolerance, Reliability, Availability

초 록 차량기지 검수고에 구조를 고려하였을 시, 철제빔으로 인해 열차를 검지하기 위한 궤도회로를 설치하기가 어렵고, 레일과 대지가 철제빔으로 연결되어 누설전류가 많아지므로 궤도회로 시스템을 적용할 수 없다. 그러므로 열차가 검수고에 들어오면 수작업으로 차량이 있다는 표시를 하여 사용하고 있으며, 잘못 취급할 시에는 열차충돌 혹은 탈선으로 이어지는 사고가 발생할 수 있다. 본 논문에서는 검수고에서 소프트웨어 프로그램을 이용하여 가상궤도회로를 적용하였다. 가상궤도회로의 알고리즘을 제시하였으며, 열차가 점유된 검수고 방향으로 신호를 취급하는 경우에 검수고 열차점유 표시 및 신호기 정지신호 표시로 인해 검수고에서 열차의 안전성을 확보하였다. 또한 프로그램의 신뢰도 및 가용도를 분석하여 소프트웨어에 적합한 결함허용 기법을 적용하였다.

주요어 : 가상궤도회로, 철도신호시스템, 소프트웨어 결함허용, 신뢰도, 가용도

1. 서 론

궤도회로 시스템은 열차의 위치 및 속도를 제어하는 시스템이다. 궤도회로 시스템은 레일에 전류를 흘려서 차축에 의해 폐루프가 형성되어 계전기가 동작함으로써 궤도점유가 되는 것을 의미한다. 또한 궤도회로 시스템은 궤도를 주파수를 이용하여 일정부분으로 나누어 전기적으로 절연을 시킨다. 차량기지 검수고에 구조를 고려하였을 시, 철제빔으로 인해 열차를 검지하기 위한 궤도회로를 설치하기가 어렵다. Fig. 1과 같이 레일과 대지가 철제빔으로 연결되어 누설전류가 많아지기 때문에 궤도회로 시스템을 적용할 수 없다[1]. 검수고에서 궤도회로 시스템을 설치할 수 없기 때문에 열차가 검수고에 들어오면 Fig. 2와 같이 수작업으로 차량이 있다는 표시를 하여 사용하고 있으며, 잘못 취급할 시에는 열차충돌 혹은 탈선으로 이어지는 사고가 발생할 수 있다[2]. 사고를 방지하기 위해서 현재 검수고에서 광센서를 이용하여 열차의 위치를 판단한다. 하지만 이 방법은 광센서의 단가가 높고, 설치구간을 차량이 입고되는 구간 및 분기구간마다 설치를 해야하므로 설치비용이 크다. 또한 기후나 환경적인 조건에 민감한 광센서는 잦은 고장발생으로 인해 유지보수 비용이 많이 든다. 그래서 철도운영기관에서도 많이 사용하고 있지 않은 실정이다.

이러한 문제점을 해결하는 하나의 방법은 가상궤도회로를 검수고에 적용하는 것이다. 가상궤도회로는 소프트웨어에 의해 실제 궤도회로가 설치되어 있는 것으로 간주하여, 열차의 진로를 제어할 수 있도록 하는 방법이다. 가상궤도회로를 검수고에 적용함으로써 열차의 안전을 보장하고, 센서 등의 설치장비 비용을 감소시킬 수 있으므로 효율적이다. 가상궤도회로는 소프트웨어에 의해 구현되므로 신뢰도 및 가용도 기법에 대한 해석이 필요하고, 가장 안전성이 높은 기

†교신저자 : 한국전기산업진흥회 연구개발팀
E-mail : kms0403@koema.or.kr
[1]유경제어㈜
[2]서울메트로 궤도신호처
[3]한국전기산업진흥회 연구개발팀
[4]서울과학기술대학교 철도전기신호공학과

Fig. 1 Depot with iron-beam

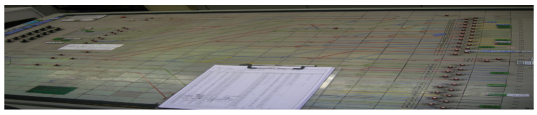


Fig. 2 Document of train number through manual labor

법으로 가상궤도회로를 적용할 필요가 있다.

본 논문에서는 검수고에서 가상궤도회로를 적용하였다. 가상궤도회로의 소프트웨어 알고리즘을 제시하였으며, 열차가 점유된 검수고 방향으로 신호를 취급하는 경우에 검수고 열차점유 표시 및 신호기 정지신호 표시로 인해 검수고에서 열차의 안전성을 확보하였다. 또한 소프트웨어의 신뢰도 및 가용도를 결함허용 기법에 따라 분석하여 소프트웨어 프로그램에 적합한 결함허용 기법을 제시하였다.

2. 가상궤도회로 알고리즘

가상궤도회로는 하드웨어적인 궤도회로시스템이 아닌 새로운 소프트웨어적 시스템이다. 검수고에서는 궤도회로의 부설이 철제빔에 의해 어려우므로 가상궤도회로를 사용하여 열차안전운행을 확보한다. 가상궤도회로의 점유설정 알고리즘은 Fig. 3과 같다[3].

가상궤도의 유무를 판단하고, 가상궤도가 있으면 가상궤도에 열차번호를 입력하여 쇄정하고 점유설정을 하도록 한다. 또한 신호현시 알고리즘은 Fig. 4와 같다.

신호현시는 안전성을 위해서 선로전환기의 쇄정 이후에 되도록 한다. Fig. 3 및 Fig. 4를 이용하여 가상궤도회로 소프트웨어에 적용한다. 적용하는 경우에 궤도회로는 전자연동

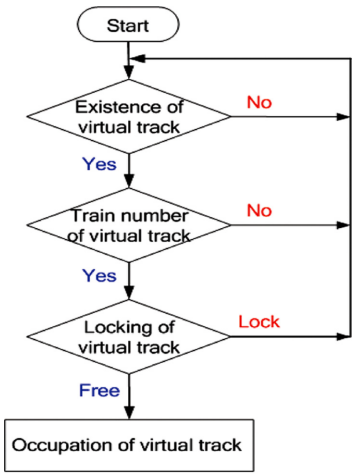


Fig. 3 Algorithm for occupation of virtual track

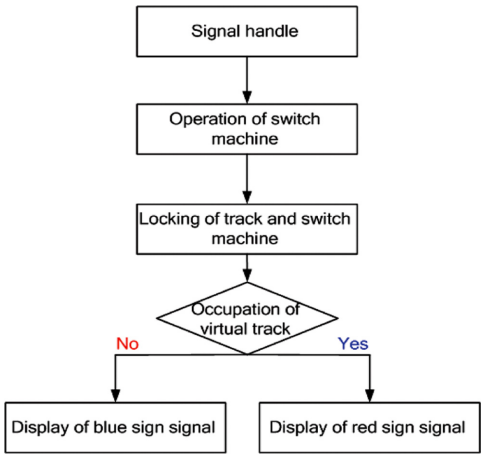


Fig. 4 Algorithm for display of signal

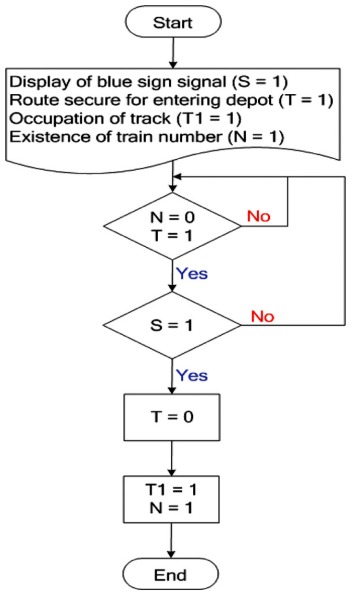


Fig. 5 Flow chart of software program in virtual track circuit

장치와의 인터페이스 부분을 고려해야 한다. 즉, 가상궤도회로를 구성할 시에 전자연동장치 데이터 베이스에 직접적인 영향을 받지 않도록 해야 한다. 이를 고려하여 프로그램 수정을 최소화할 필요가 있다.

Fig. 3과 Fig. 4를 적용하여 가상궤도회로의 프로그램 순서도를 나타내면 Fig. 5와 같다.

3. 소프트웨어 결함허용기법

결함허용은 소프트웨어에 결함이 존재하는 것은 불가피하다고 하고, 결함의 발생 방법과 외부의 영향 현상법에 착안하여 대처하는 것이다[4].

소프트웨어에서도 하드웨어와 시스템의 분야 에서의 경우와 마찬가지로 이중화성의 개념을 도입 하여 결함허용을 실현하는 기술이 있다. 구체적 으로는 소정의 기능을 제공하

는 소프트웨어를 복수 구성으로 하고, 결함이 발생되었을 경우에 다수결 논리로 정정하기도 하고, 대체 소프트웨어로 교체 하는 것에 의해 외부에는 결함이 발생하지 않은 것처럼 보인다. 소프트웨어 결함허용 기법에는 복구블록, N-버전 프로그래밍, *N* 자기검사 프로그래밍이 있다[5,6].

3.1 복구블록

복구 블록은 Fig. 6에서와 같이 주 처리블록이 동작하고 있는 상태를 상태 0으로, 주 처리블록이 고장 나고 첫 번째 대체 블록이 동작하는 상태를 상태 1로 한다. 또한 *N*-1번째 대체 처리블록이 동작하는 상태를 상태 *N*-1로 모델링이 이루어진다. 그리고 모두 고장이 난 상태를 *F*로 표시하며, Fig. 6의 모델을 수리적으로 해석하여 각각 상태에서 의 확률은 식(1), 식(2)와 같다[7].

$$R_i(t) = \lambda_{PE}^i \frac{1}{i!} t^i e^{-(\lambda_{PE}+\lambda_{AE})t} \quad (i = 0,..., N\text{-}1) \tag{1}$$

$$F_i(t) = 1 - \sum_{t=0}^{N-1} \lambda_{PE}^i \frac{1}{i!} t^i e^{-(\lambda_{PE}+\lambda_{AE})t} \tag{2}$$

*N*은 대체블록의 수를 의미한다. λ_{PE}는 프로그램 결함발생률을 의미하고, λ_{AE}는 적응검사의 결함 발생률을 의미한다.

복구블록의 신뢰도는 상태 0에서 *N*-1까지 있을 확률이므로 식(3)과 같이 계산한다.

$$R_{N-1}(t) = 1 - \sum_{t=0}^{N-1} \lambda_{PE}^i \frac{1}{i!} t^i e^{-(\lambda_{PE}+\lambda_{AE})t} \tag{3}$$

Fig. 6을 이용하여 마르코프 모델에서 각 상태의 확률은 식(4)와 같다.

$$P = \begin{bmatrix} 0 & 1 & \cdots & N-1 & F \\ 0 & \lambda_{PE} & \cdots & 0 & 0 \\ \mu_{PE} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \lambda_{PE}+\lambda_{AE} \\ 0 & 0 & \cdots & \mu_{PE}+\mu_{AE} & 0 \end{bmatrix} \tag{4}$$

μ_{PE}는 프로그램 수리율을 의미하고, μ_{AE}는 적응검사의 수리율을 의미한다. 식(4)을 이용하여 마르코프 미분 방정식을 행렬로 나타내면 식(5)와 같다.

$$\begin{bmatrix} P_0'(t) \\ P_1'(t) \\ \vdots \\ P_{N-1}'(t) \\ P_F'(t) \end{bmatrix} = \begin{bmatrix} -\lambda_{PE} & \mu_{PE} & \cdots & 0 & 0 \\ \lambda_{PE} & -(\lambda_{PE}+\mu_{PE}) & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & -(\lambda_{PE}+\lambda_{AE}+\mu_{AE}) & \mu_{PE}+\mu_{AE} \\ 0 & 0 & \cdots & \lambda_{PE}+\lambda_{AE} & -(\mu_{PE}+\mu_{AE}) \end{bmatrix} \begin{bmatrix} P_0(t) \\ P_1(t) \\ \vdots \\ P_{N-1}(t) \\ P_F(t) \end{bmatrix} \tag{5}$$

식(5)에서 정상상태 확률을 계산하면 식(6)과 같다.

$$P = \begin{bmatrix} 1-\lambda_{PE} & \lambda_{PE} & \cdots & 0 & 0 \\ \mu_{PE} & 1-(\lambda_{PE}+\mu_{PE}) & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1-(\lambda_{PE}+\lambda_{AE}+\mu_{AE}) & \lambda_{PE}+\lambda_{AE} \\ 0 & 0 & \cdots & \mu_{PE}+\mu_{AE} & 1-(\mu_{PE}+\mu_{AE}) \end{bmatrix} \tag{6}$$

각 상태에서의 확률을 계산하면 식(7)과 같다.

$$\begin{aligned} P_0 &= (1-\lambda_{PE})P_0 + \mu_{PE}P_1 + \cdots P_{N-1} \\ P_1 &= \lambda_{PE}P_0 + (1-\lambda_{PE}-\mu_{PE})P_1 + \cdots P_{N-1} \\ &\vdots \qquad\qquad \vdots \\ P_{N-1} &= \cdots + (1-\lambda_{PE}-\lambda_{AE}-\mu_{AE})P_{N-1} + (\mu_{PE}+\mu_{AE})P_F \\ P_F &= \cdots + (\lambda_{PE}+\lambda_{AE})P_{N-1} + (1-\mu_{PE}-\mu_{AE})P_F \end{aligned} \tag{7}$$

식(7)에 "모든 각 상태에서의 확률을 더한 결과는 1이다" 라는 조건을 삽입하여 고장 전까지 의 상태인 *N*-1까지 가용도를 계산하면 식(8)과 같다.

$$A = p_0 + p_1 + \cdots + p_{N-1} \tag{8}$$

3.2 *N*-버전 프로그래밍

N-버전 프로그래밍은 Fig. 7처럼 *N*개의 버전 모두가 올바르게 동작하고 있는 상태를 상태 *N*으로, (*N*+1)/2개의 버전이 동작할 때까지 진행되다가 시스 템 고장 상태인 상태 *F*에 도달하게 된다[7].

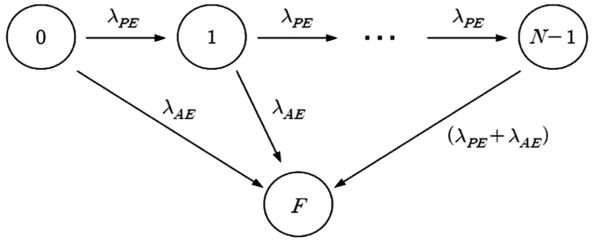


Fig. 6 Recovery block

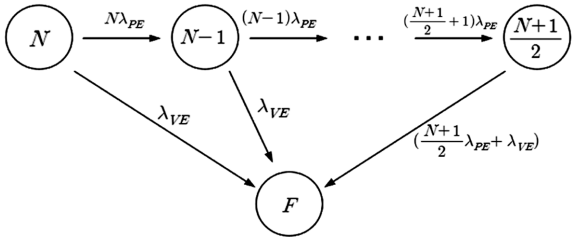


Fig. 7 N-version programming

Fig. 7과 같은 마르코프 모델을 수리적으로 해석하여 각 각 상태에서의 확률은 식(9), 식(10), 식(11)과 같다.

$$R_N(t) = e^{-(N\lambda_{PE}+\lambda_{VE})t} \tag{9}$$

$$R_{N-1}(t) = N(e^{-((N-1)\lambda_{PE}+\lambda_{VE})t} - e^{-(N\lambda_{PE}+\lambda_{VE})t}) \tag{10}$$

$$R_{N-2}(t) = \frac{N(N-1)}{0}(e^{-((N-2)\lambda_{PE}+\lambda_{VE})t} - 2e^{-((N-1)\lambda_{PE}+\lambda_{VE})t} + e^{-(N\lambda_{PE}+\lambda_{VE})t}) \tag{11}$$

λ_{VE}는 보터의 결함발생률을 의미한다. Fig. 7을 이용하여 마르코프 모델에서 각 상태의 확률은 식(12)와 같다.

$$p = \begin{bmatrix} 0 & 1 & \cdots & N-1 & F \\ 0 & N\lambda_{PE} & \cdots & 0 & 0 \\ N\mu_{PE} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \frac{N+1}{2}\lambda_{PE}+\lambda_{VE} \\ 0 & 0 & \cdots & \frac{N+1}{2}\mu_{PE}+\mu_{VE} & 0 \end{bmatrix} \tag{12}$$

μ_{VE}는 보터의 수리율을 의미한다. 식(12)를 이용하여 마르코프 미분 방정식을 행렬로 나타내면 식(13)과 같다.

$$\begin{bmatrix} P_0'(t) \\ P_1'(t) \\ \vdots \\ P_{N-1}'(t) \\ P_F'(t) \end{bmatrix} = \begin{bmatrix} -N\lambda_{PE} & N\mu_{PE} & \cdots & 0 & 0 \\ N\lambda_{PE} & -((N-1)\lambda_{PE}+N\mu_{PE}) & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & -\left(\frac{N+1}{2}\lambda_{PE}+\lambda_{VE}+\mu_{VE}\right) & \frac{N+1}{2}\mu_{PE}+\mu_{VE} \\ 0 & 0 & \cdots & \frac{M+1}{2}\lambda_{PE}+\lambda_{VE} & -\left(\frac{N+1}{2}\mu_{PE}+\mu_{VE}\right) \end{bmatrix} \begin{bmatrix} P_0(t) \\ P_1(t) \\ \vdots \\ P_{N-1}(t) \\ P_F(t) \end{bmatrix} \tag{13}$$

식(13)에서 정상상태 확률을 계산하면 식(14)와 같다.

$$P = \begin{bmatrix} 1-N\lambda_{PE} & N\lambda_{PE} & \cdots & 0 & 0 \\ N\mu_{PE} & 1-((N-1)\lambda_{PE}+N\mu_{PE}) & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1-\left(\frac{N+1}{2}\lambda_{PE}+\lambda_{VE}+\mu_{VE}\right) & \frac{N+1}{2}\lambda_{PE}+\lambda_{VE} \\ 0 & 0 & \cdots & \frac{M+1}{2}\mu_{PE}+\mu_{VE} & 1-\left(\frac{N+1}{2}\mu_{PE}+\mu_{VE}\right) \end{bmatrix} \tag{14}$$

각 상태에서의 확률을 계산하면 식(15)와 같다.

$$\begin{aligned} P_0 &= (1-N\lambda_{PE})P_0 + N\mu_{PE}P_1 + \cdots P_{N-1} \\ P_1 &= N\lambda_{PE}P_0 + (1-(N-1)\lambda_{PE}-N\mu_{PE})P_1 + \cdots P_{N-1} \\ &\vdots \qquad\qquad \vdots \\ P_{N-1} &= \cdots + \left(1-\frac{N+1}{2}\lambda_{PE}-\lambda_{VE}\mu_{VE}\right)P_{N-1} + \left(\frac{N+1}{2}\mu_{PE}+\mu_{VE}\right)P_F \\ P_F &= \cdots + \left(\frac{N+1}{2}\lambda_{PE}+\lambda_{VE}\right)P_{N-1} + \left(1-\frac{N+1}{2}\mu_{PE}-\mu_{VE}\right)P_F \end{aligned} \tag{15}$$

가용도를 계산하기 위해 식(15)을 식(8)에 대입 한다.

3.3 N 자기검사 프로그래밍

N 자기검사 프로그래밍은 하드웨어 결함허용의 동적 중복에 해당하는 것으로, 실행 중에 자신의 동적 동작을 검사하는 자기검사 프로그램을 이용 하여 결함허용 구조를 만든 것이다. *N* 자기검사 프로그래밍은 Fig. 8과 같이 모든 버전이 올바르게 동작하고 있는 상태를 상태 0으로 하나의 버전이 고장이 나서 두 번째 버전이 동작하는 상태를 상태 1로 한다. 마지막 버전이 동작하는 상태를 상태 *N*-1로 나타내고 모든 버전이 고장이 난 상태를 상태 *F*로 표시한다.

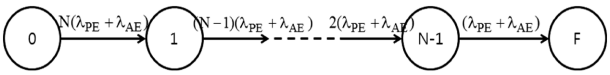


Fig. 8 N self checking programming

Fig. 8의 마르코프 모델에서 각각 상태에서의 확률은 식(16), 식(17), 식(18)과 같다.

$$R_0(t) = e^{-N(\lambda_{PE}+\lambda_{AE})t} \tag{16}$$

$$R_1(t) = N(e^{-(N-1)(\lambda_{PE}+\lambda_{AE})t} - e^{-N(\lambda_{PE}+\lambda_{AE})t}) \tag{17}$$

$$R_{N-1}(t) = \frac{N(N-1)}{2}(e^{-(N-2)(\lambda_{PE}+\lambda_{AE})t} - 2e^{-(N-1)(\lambda_{PE}+\lambda_{AE})t} + e^{-N(\lambda_{PE}+\lambda_{AE})t}) \tag{18}$$

Fig. 8을 이용하여 마르코프 모델에서 각 상태의 확률은 식(19)와 같다.

$$P = \begin{bmatrix} 0 & 1 & \cdots & N-1 & F \\ 0 & N(\lambda_{PE}+\lambda_{AE}) & \cdots & 0 & 0 \\ N(\mu_{PE}+\mu_{AE}) & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \lambda_{PE}+\lambda_{AE} \\ 0 & 0 & \cdots & \mu_{PE}+\mu_{AE} & 0 \end{bmatrix} \tag{19}$$

식(19)를 이용하여 마르코프 미분 방정식을 행렬로 나타내면 식(20)과 같다.

$$\begin{bmatrix} P_0'(t) \\ P_1'(t) \\ \vdots \\ P_{N-1}'(t) \\ P_F'(t) \end{bmatrix} = \begin{bmatrix} -N(\lambda_{PE}+\lambda_{AE}) & N(\mu_{PE}+\mu_{AE}) & \cdots \\ N(\lambda_{PE}+\lambda_{AE}) & -(N(\lambda_{PE}+\mu_{PE})+(N-1)(\lambda_{PE}+\mu_{PE})) & \cdots \\ \vdots & \vdots & \cdots \\ 0 & 0 & \cdots \\ 0 & 0 & \cdots \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 1-3(\lambda_{PE}+\lambda_{AE}+\mu_{PE}+\mu_{AE}) & \mu_{PE}+\mu_{AE} \\ \lambda_{PE}+\lambda_{AE} & 1-(\mu_{PE}+\mu_{AE}) \end{bmatrix} \begin{bmatrix} P_0(t) \\ P_1(t) \\ \vdots \\ P_{N-1}(t) \\ P_F(t) \end{bmatrix} \quad (20)$$

식(20)에서 정상상태 확률을 계산하면 식(21)과 같다.

$$P = \begin{bmatrix} 1-N(\lambda_{PE}+\lambda_{AE}) & N(\lambda_{PE}+\lambda_{AE}) & \cdots \\ N(\mu_{PE}+\mu_{AE}) & 1-(N(\lambda_{PE}+\mu_{PE})+(N-1)(\lambda_{PE}+\mu_{PE})) & \cdots \\ \vdots & \vdots & \cdots \\ 0 & 0 & \cdots \\ 0 & 0 & \cdots \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 1-3(\lambda_{PE}+\lambda_{AE}+\mu_{PE}+\mu_{AE}) & \lambda_{PE}+\lambda_{AE} \\ \mu_{PE}+\mu_{AE} & 1-(\mu_{PE}+\mu_{AE}) \end{bmatrix} \quad (21)$$

각 상태에서의 확률을 계산하면 식(22)와 같다.

$$\begin{aligned} P_0 &= (1-N(\lambda_{PE}+\lambda_{AE}))P_0+N(\mu_{PE}+\mu_{AE})P_1+\cdots P_{N-1} \\ P_1 &= N(\lambda_{PE}+\lambda_{AE})P_0+(1-(N(\lambda_{PE}+\lambda_{AE}))+(N-1)(\lambda_{PE}+\lambda_{AE}))P_1+\cdots P_{N-1} \\ &\vdots \qquad\qquad \vdots \\ P_{N-1} &= \cdots+(1-3(\lambda_{PE}+\lambda_{AE}+\mu_{PE}+\mu_{AE}))P_{N-1}+(\mu_{PE}+\mu_{AE})P_E \\ P_F &= \cdots+(\lambda_{PE}+\lambda_{AE})P_{N-1}+(1-\mu_{PE}-\mu_{AE})P_F \end{aligned} \quad (22)$$

가용도를 계산하기 위해 식(22)를 식(8)에 대입한다.

4. 시뮬레이션

4.1 가상궤도회로 구현

제시한 가상궤도회로 알고리즘을 이용하여 윈도우 환경에서 프로그램을 작성한 가상궤도 점유 표시 순서는 Fig. 9와 같다.

Fig. 9의 가상궤도 점유 순서는 먼저 가상궤도 방향으로 신호를 취급한 후, 가상궤도 방향으로 열차가 진입한다. 열차가 진입하면 적색신호를 현시하며 열차번호가 부여되고, 또한 열차진로가 모두 점유로 표시된다. 이를 통해 열차가 안전하게 검수고에 진입할 수 있다. 가상궤도회로의 적용전과 후를 비교하면 Fig. 10과 같다.

가상궤도회로 적용 전에는 열차가 점유된 검수고 방향으로 신호 취급 시에 신호가 현시된다. 그래서 검수고 열차 무표시 및 신호기 진행표시로 열차의 충돌 위험이 많다. 그러나 가상궤도회로 적용후에는 열차가 점유된 검수고 방향으로 신호 취급 시에 정지신호가 현시되므로 열차의 충돌 위험이 없다.

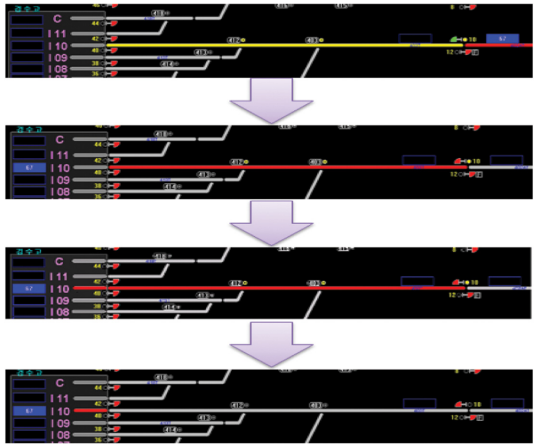
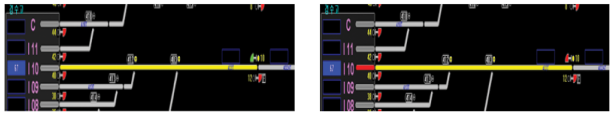


Fig. 9 Occupation of virtual track



(a) Before (b) After

Fig. 10 Before and after application of virtual track

4.2 소프트웨어 결합허용기법에 의한 신뢰도 및 가용도 분석

4.2.1 신뢰도 분석

가상궤도회로의 프로그램 결함발생 (10^{-6}/시간), 적응검사의 결함 발생 (10^{-8}/시간), 보터의 결함발생 (10^{-8}/시간)의 결함발생률 λ는 고정하고, 시간을 변화하면서 신뢰도를 분석하였다[8,9]. Fig. 11~Fig. 13은 소프 트웨어 결함허용기법에서의 신뢰도를 나타낸다.

같은 수의 대체블록이 있는 경우에는 복구블록 기법이 N-버전 프로그래밍 및 N 자기검사 프로그래밍 기법보다 신뢰도가 높은 것을 확인할 수 있다. 또한 N-버전 프로그래밍 및 N 자기검사 프로그래밍 기법은 시간이 지날수록 천천히 신

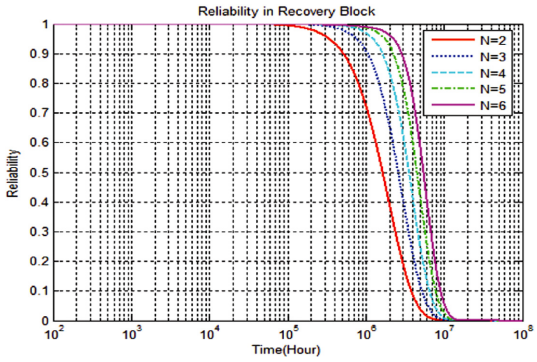


Fig. 11 Reliability of recovery block

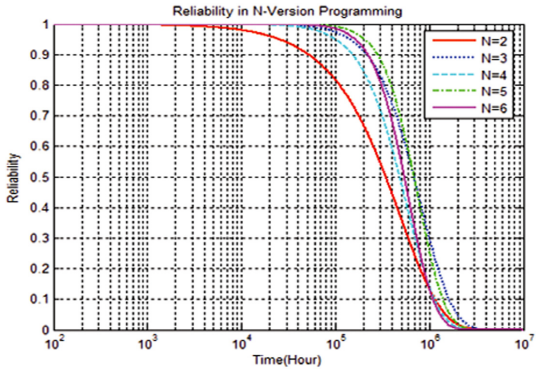


Fig. 12 Reliability of N-version programming

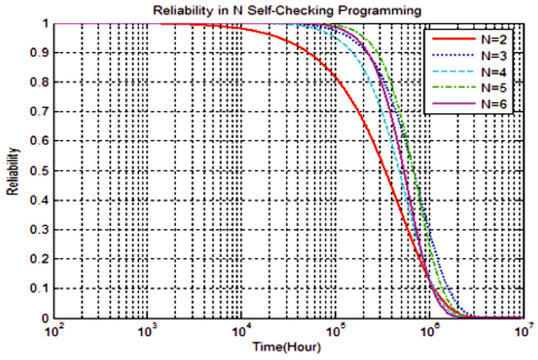


Fig. 13 Reliability of N self checking programming

뢰도가 감소하는 반면에 복구블록 기법은 일정시간 이후에 신뢰도가 급격히 감소하는 것을 확인할 수 있다. 복구블록 기법은 대체블록 수가 높아질수록 신뢰도가 높은 반면에 *N*-버전 프로그래밍 및 *N* 자기검사 프로그래밍 기법은 홀수개의 버전이 증가될 때, 더 높은 신뢰도를 얻을 수 있다.

4.2.2 가용도 분석

가상궤도회로의 프로그램 결함복구율 (0.05시간), 적응검사의 결함복구율 (0.05시간), 보터의 결함 복구율 (0.05시간)의 μ는 고정하고, 정상상태 의 시간에서 가용도를 분석하였다 [8,9]. Table 1~Table 3은 소프트웨어 결함허용기법에서의 가용도를 나타낸다.

같은 수의 대체블록이 있는 경우에 신뢰도와 같이 복구블록, *N* 자기검사 프로그래밍, *N*-버전 프로그래밍 기법 순으로 가용도가 높다. 대체블록의 수가 증가할수록 가용도가 증가하지만 *N*-버전 프로그래밍의 경우에는 신뢰도와 마찬가지

Table 1 Availability of recovery block

The number of recovery block	Availability
2	0.999999999798
3	1
4	1
5	1
6	1

Table 2 Availability of N-version programming

The number of *N*-version programming	Availability
2	0.99998660017956
3	0.99999999973201
4	0.99999999969901
5	0.99999999999999
6	0.99999999999989

Table 3 Availability of N self checking programming

The number of *N* self checking programming	Availability
2	0.99999999989799
3	0.99999999999999
4	1
5	1
6	1

로 홀수 버전 프로그래밍이 짝수 버전 프로그래밍보다 가용도가 더 높게 해석되었다. 가용도 분석결과, 대체블록이 2개 이상인 경우에 모든 기법에서 가용도가 99.9% 이상으로 높게 해석되었다.

5. 결 론

본 논문에서는 가상궤도회로의 알고리즘을 제시하 였으며, 열차가 점유된 검수고 방향으로 신호를 취급하는 경우에 검수고 열차점유 표시 및 신호기 정지신호 표시로 인해 검수고에서 열차의 안전성을 확보하였다. 또한 프로그램의 신뢰도 및 가용도를 분석하여 소프트웨어 프로그램에 적합한 결함허용 기법을 제시하였다. 가상궤도회로에서 열차가 진입하는 동안 열차번호가 부여되면서 가상궤도 회로의 진로를 점유로 표시한다. 열차가 검수고에 모두 진입한 후에는 열차진로에 해당하는 궤도를 비점유로 표시하므로 열차 운행 안전성을 높일 수 있으며 또한 예산절감의 효과를 크게 볼 수 있다. 또한 소프트웨어를 이용하여 가상궤도회로 시스템을 설계할 경우, 소프트웨어 결함허용기법들의 해석 결과는 다음과 같다.

(1) 결함허용기법에서 같은 수의 대체블록이 있는 경우에는 복구블록 기법이 *N*-버전 프로그래밍 및 *N* 자기검사 프로그래밍 기법보다 신뢰도 및 가용도가 높다.

(2) 신뢰도가 1로 유지되는 시간을 기준으로 보면, 복구블록 기법이 *N*-버전 프로그래밍 및 *N* 자기검사 프로그래밍 기법보다 약 10배 많다. 즉, 신뢰도가 높다. 또한 가용도도 복구블록 기법이 가장 높다.

(3) 복구블록 기법은 일정시간 이후에 급격히 신뢰도가 감소하고, *N*-버전 프로그래밍 및 *N* 자기검사 프로그래밍 기법은 시간이 지날수록 천천히 신뢰도가 감소한다.

(4) 복구블록 기법은 대체블록 수가 높아질수록 신뢰도가 높은 반면에 *N*-버전 프로그래밍 및 *N* 자기검사 프로그래밍 기법은 짝수 버전 프로그래밍 보단 홀수 버전 프로그래밍이 더 높은 신뢰도 및 가용도를 가진다. 즉, *N*-버전 프로그래밍 및 *N* 자기검사 프로그래밍 기법에는 짝수보단 홀수개의 버전이 증가될 때, 더 높은 신뢰도 및 가용도를 가진다.

복구블록기법이 가상궤도회로 시스템 설계시에 신뢰도 및 가용도가 가장 높게 해석되었다. 본 연구는 프로그램 결함발생, 적응검사의 결함발생, 보터의 결함발생의 결함발생률과 수리율은 고정하고, 시간을 변화하면서 신뢰도 및 가용도를 분석하였 는데, 결함발생률 및 수리율은 시간에 따라 증가될 수 있으므로 향후 더 정확한 해석을 위해 가변함수로 해석할 필요가 있다. 그리고 신뢰도 및 가용도 뿐만 아니라 안전도, 유지보수 등을 고려하여 가상궤도회로가 체계적이고 안정적인 설계가 이루어져야 한다.

후 기

본 연구는 서울메트로의 지원하에 “가상궤도회로개발” 일환으로 수해되었으며, 이에 관계자 여러분께 감사드립니다.

참고문헌

[1] Seoul metro (2009) Installation of Virtual Track Circuit in Depot, pp. 3-5.

[2] J.S. Kim (2010) Safety Management for Operation in Depot, Seoul metro, pp. 8-15.

[3] Yookyung Control Co. Ltd (2010) Software Algorithm for Virtual Track Circuit, pp. 1-3.

[4] J.W. Lee (1999) Signal Control System Engineering Research & Development, *Proceedings of the Korean Institute of Electrical Engineers*, pp. 51-57.

[5] G.h. Min, J.W. Lee (2008) A Study on a Safety Activity on Safety Critical Related Software in Train Control System, *Proceedings of the Korean Society of Automotive Engineers, Annual Conference & Exhibition*, 2(1), pp. 1077-1083.

[6] J.K. Hwang (2000) Software Design as Simulator in Signal Control System, *Proceedings of the Korean Society for Railway*, pp. 269-275.

[7] S.C. Suh, J.W. Lee (2009) Reliability Analysis for Train Control System by Software Fault Tolerance Techniques, *Journal of the Korean Society for Railway*, 12(6), pp. 1043-1048.

[8] J.B. Dugan, M.R. Lyu (1995) Dependability Modeling for Fault-Tolerant Software and Systems Software Fault Tolerance, Wiley Trends in Software Book Series, Wiley, pp. 109-138.

[9] K.H. Kim, H.O. Welch (1989) Distributed Execution of Recovery Blocks : An Approach for Uniform Treatment of Hardware and Software Faults in Real-Time Applications, *IEEE Transactions on Computers*, 38(5), pp.625-636.

접수일(2011년 11월 10일), 게재확정일(2012년 2월 17일)