Journal of Institute of Control, Robotics and Systems (2015) 21(3):187-193
http://dx.doi.org/10.5302/J.ICROS.2015.14.9010 ISSN:1976-5622 eISSN:2233-4335

자동차 제어 시스템의 실시간 성능 검증을 위한 효율적인 실시간 시뮬레이션 기법

An Efficient Simulation Technique to Verify Real-time Performance of Vehicle Control Systems

김 승 곤, 위 경 수, 이 창 건*, 이 경 수
(Seunggon Kim[1], Kyoung-Soo We[1], Chang-Gun Lee[1,*], and Kyongsu Yi[2])
[1]Department of Computer Science and Engineering, Seoul National University
[2]Department of Mechanical and Aerospace Engineering, Seoul National University

Abstract: When developing a vehicle control system, simulation methods are widely used to validate the whole system in the early development phase. With this regard, the simulator should correctly behave just like the real parts that are not yet implemented while interacting with already implemented parts in real-time. However, most simulators cannot provide functionally and temporally accurate behaviors of the target system. In order to overcome this limitation, this paper proposes a novel real-time simulation technique that can efficiently simulate the temporal behavior as well as the functional behavior of the simulation target system.

Keywords: vehicle control system, real-world interaction, simulation correctness, task modeling, timing constraints

I. 서론

무인 자율 주행 자동차 등의 자동차 제어 시스템과 같이 크고 복잡한 내장형 컴퓨팅 시스템이 개발될 때 실시간 성능 검증을 위하여 시뮬레이션 기법이 이용된다[1]. 현대 자동차의 경우, 서로 다른 5개의 네트워크에 연결된 70개 이상의 ECU (Electronic Control Unit)를 가지기 때문에 복잡한 구조로 인하여 시뮬레이션을 이용한 검증 없이 한 번에 개발하는 것은 어렵다[2]. 여기에서 언급하는 시뮬레이션이란 대상 컴퓨팅 시스템 및 해당 시스템을 포함하는 네트워크와 네트워크에 연결된 다른 컴퓨팅 시스템 위에서 수행될 일부 혹은 모든 태스크들의 실시간 행태를 PC (Personal Computer) 등의 범용 컴퓨팅 장비에서 모사하는 것을 의미한다. 그러나 시중의 "실시간 시뮬레이션"이라 칭하는 시뮬레이터는 단지 실세계를 모사하기 위한 기능성 검증에 치중할 뿐 타이밍 검증은 전혀 고려되지 않는다. 그렇기 때문에 이미 실제로 개발되어진 시스템과 함께 시뮬레이션 할 때 상호작용이 제대로 이루어지지 않아 정확하지 않은 결과가 나올 수 있다.

이러한 문제를 해결하기 위한 방법을 제안하기에 앞서, 설명의 편의를 위해 그림 1과 같은 시스템을 가정한다. 그림 1은 네트워크로 연결된 3개의 ECU {E_1, E_2, E_3}와 그 위에서 수행되는 5개의 태스크 {τ_1, ..., τ_5}로 이루어진 자동차 제어 시스템의 한 예를 나타낸다. 각 ECU는 전용의 네트워크 선인 CAN (Controller Area Netwrok) 버스로 연결되어 있다고 가정한다. 여기에서 E_3는 실제로 구현되어진 ECU인 반면, E_1과 E_2는 시뮬레이션 되어져야 한다. 이에 따라 E_1, E_2의 위에서 수행되는 태스크 {τ_1, τ_2, τ_3, τ_4, τ_5} 또한 시뮬레이션 될 대상이다. 시뮬레이션 될 대상 시스템을 "시뮬레이션 시스템 (Simulated-System)", 시뮬레이션 시스템 내의 각 ECU에서 실행되는 태스크를 "시뮬레이션 태스크(Simulated-Task)"라 하자. 또한 이미 실제로 구현되어있는 ECU는 "실제 ECU (Real-ECU)"라 하자. 이때 시뮬레이션 ECU와 실제 ECU 사이에서 발생하는 데이터 교환 등의 상호작용을 "실세계 상호작용 (Real-world Interaction)"이라 하자. 여기에서 시뮬레이션이란 시뮬레이션 태스크들이 각각의 ECU에서 수행되었을 때의 행태를 PC 등의 범용 컴퓨팅 장비에서 모사하는 것을 의미한다.

그림 1과 같은 환경에서 특정 순간에 시뮬레이션 태스크들이 그림 2(a)와 같은 실시간 행태를 보였다고 가정하자. 그림에서 태스크 수행 사이의 화살표는 데이터 교환을 의미한다. 그림 2(a)에서 각 태스크들은 자신의 수행 시작 시점에 데이터를 받아서 처리한 뒤 수행이 종료되면 데이터를 내보낸다고 가정한다. 태스크 간 데이터 교환 중 실제 ECU와 시뮬레이션 ECU간 데이터 교환이 일어나는 시점을 "실세계 상호작용 시점(Real-world Interaction Point)"라

* Corresponding Author
Manuscript received November 15, 2014 / revised December 15, 2014 / accepted December 30, 2014
김승곤, 위경수, 이창건: 서울대학교 컴퓨터 공학부
(hexoul@snu.ac.kr/we123456@snu.ac.kr/cglee@snu.ac.kr)
이경수: 서울대학교 기계항공공학부(kyi@snu.ac.kr)
※ 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송연구개발사업[200352423, CPS를 위한 콤포넌트 기반 설계이론 및 제어커널 개발]과 지식경제부 및 한국산업기술진흥원의 국제공동기술개발사업(M002300089)의 연구결과로 수행되었음. 이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.

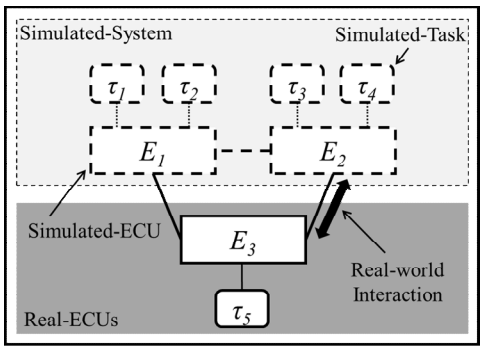


그림 1. 자동차 제어 시스템 예제.

Fig. 1. Example vehicle control system.

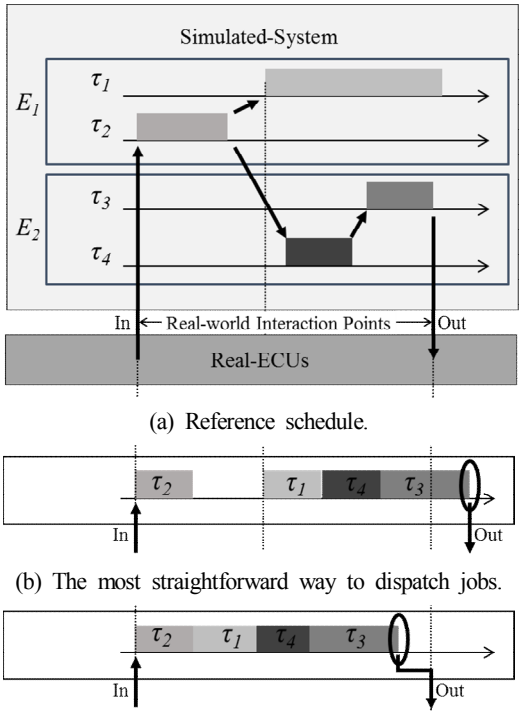


그림 2. 그림 1 시스템의 실시간 행태 및 모사 유형.

Fig. 2. A motivating example on Fig. 1.

고 하자. 이때 모든 태스크의 종료시점과 실세계 상호작용 시점을 관찰함으로써 데이터가 그림 2(c)와 같이 정확한 시점에 전달되어 졌는지 확인할 수 있다. 예상 행태에 따라 제대로 모사되어졌다면 시뮬레이션이 제대로 이루어졌다고 볼 수 있다.

시뮬레이션을 행할 시뮬레이터의 범용 컴퓨팅 장비는 PC로 시뮬레이터를 동작시키는 호스트 프로세서는 시뮬레이션 호스트 (Simulation Host)로 부르자. 시뮬레이션 호스트는 싱글 코어 (Single-Core) 라고 가정한다. 시뮬레이션 호스트의 연산 성능은 실제 ECU보다 뛰어나다고 가정한다. 이때, 그림 2(b)는 시뮬레이션 호스트가 예상 행태에서 태스크의 시작 시간에 맞춰 태스크를 실행시킴으로써 가장 간단하게 모사하는 행태를 보인다. 그러나 이러한 방법은 그림 2(b)에서 보이는 것과 같이 τ_3의 수행이 늦어져 실세계 상호작용 시점을 지키지 못할 수도 있기 때문에 항상 가능한 방법은 아니다. 그러나 만약 τ_1를 τ_2의 수행이 끝난 직후에 시작했었다면 그림 2(c)와 같이 실세계 상호작용 시점이 만족되었을 것이다. 이러한 점은 시뮬레이터가 데이터 의존도를 유지시키면서도 모든 시뮬레이션 태스크들에 대하여 예상 종료 시점보다 앞서 수행을 마칠 수 있다는 점을 시사한다. 따라서 시뮬레이터는 정확한 시점에 정확한 결과를 실제 ECU에 내보내는 것이 가능하다. 이와 같이 많은 수의 시뮬레이션 ECU 및 시뮬레이션 태스크들을 모사할 때 정확성과 효율성을 모두 고려해야 한다.

그러나 개발용으로 사용되고 있는 많은 시뮬레이터들은 기능성 검증에 치중하여 실제로 작업이 수행될 환경을 고려하지 않고 최대한의 자원을 이용하여 시뮬레이션하거나 시뮬레이션 수행 시간을 줄이는 것에 집중하고 있다. 따라서 각 태스크들이 언제 시작되고 끝나는지를 보고 시간적 정확성을 검증할 수 있는 부분이 전혀 없거나[3], 시뮬레이션 호스트와 ECU 간의 성능 차이를 고려하지 않아 실시간 행태를 정확하게 모사하지 못한다[4]. 이에 따라 본고는 시뮬레이션 ECU와 실제 ECU 사이에 정확한 상호작용을 보장하는 효율적인 시뮬레이션 기법을 제안한다.

II. 관련 연구

시뮬레이션은 어떠한 실제 하드웨어 플랫폼도 없는 개발 초기에 소프트웨어 시스템들을 검증하기 위해서 널리 사용된다. 전형적으로, 소프트웨어 시스템은 명령어 집합 시뮬레이터를 이용하여 시뮬레이션 된다. 비록 이러한 시뮬레이터들이 단계별 정확도를 제공한다 하더라도, 그것들은 많은 임베디드 프로세서를 포함한 대형 시스템을 시뮬레이션 하기는 너무 느리고 실시간 시뮬레이션으로 쓸 수 없다. 시뮬레이션 속도를 증진시키기 위해서 이진 변환을 이용하는 가상 플랫폼 시뮬레이터도 있다[5,6]. 그러나 이러한 시뮬레이터는 오직 빠르고 기능적인 시뮬레이션에만 집중되어있다. 그래서 시간적 시뮬레이션 정확성에 대해서는 고려되지 않고 있다. 최근 호스트 컴파일 시뮬레이션은 빠르고 시간적으로 정확한 시뮬레이션 때문에 많은 관심을 끌고 있다[7,8]. 그러나 이것은 스마트 폰과 같이 관제실과의 어떠한 상호작용 없이 독립된 장치만 대상으로 하고 있다. 따라서 이것은 정확한 실세계와의 상호작용을 고려하지 않는다. [9]의 저자는 RTOS 스케줄링 이벤트를 시뮬레이션 하여 작업을 수행하는 RTOS 시뮬레이터를 제안하였다. 그러나 시뮬레이션 정확성은 명확하게 논의되지 않았고 하나의 RTOS 임베디드 프로세서에 제한되어 있었다. 게다가 [9]의 저자는 시뮬레이션 된 작업의 수행을 위한 어떠한 구체적인 알고리즘도 제공하지 않았다.

심지어 “실시간 시뮬레이션”이라 광고하는 상업용 시뮬레이터도 관제실과 동시에 시뮬레이션 되는 것만 제공한다. 그리고 태스크 스케줄링, 선점, 버스 중재 등과 같은 대상 시스템에서 발생하는 이벤트들 때문에 시간적 행태에 대하여 정확한 모델을 가지지 못한다[10,11]. 그렇기 때문에 시뮬레이션은 실세계 상호작용 관점에서 보았을 때 부정확하다.

III. 문제 기술

본고에서는 시뮬레이션 호스트 H가 n개의 ECU {E_1, E_2, ..., E_n}와 그 위에서 실행될 m개의 태스크 {τ_1, τ_2, ..., τ_n}를 가지는 상황을 가정한다. 이 때 각 태스크 τ_i는 다음과 같이 기술된다.

$$\tau_i = (F_i, P_i, C_i, C'_i)$$

여기에서 F_i는 목표 ECU 및 시뮬레이션 호스트에서 모두 컴파일 가능한 일련의 코드 또는 함수이고, P_i는 주기이다. C_i는 ECU에서의 최악 수행 시간이고 C'_i는 시뮬레이션 호스트에서의 최악 수행 시간이다. 주목해야할 점은 일반적으로 시뮬레이션 호스트 H의 연산 능력이 ECU보다 뛰어나므로 C_i가 C'_i보다 작다는 점이다. 주기적 태스크 τ_i의 j번째 작업(Job)은 $J_{i,j}$로 표현한다. τ_j가 τ_i의 결과를 이용하는 경우 다음과 같이 표현될 수 있다.

$$\tau_i \rightarrow \tau_j$$

위와 같이 데이터 전달 관계를 가질 때, τ_j가 τ_i의 후임자, τ_i가 τ_j의 선임자라고 하자. 전임자의 출력을 입력으로 받아 수행하는 태스크는 전임자로부터 받은 데이터 중 최신의 데이터만을 입력으로써 이용한다고 가정한다. 또한 태스크가 완전히 수행을 종료해야지만 후임자에게 수행 결과가 데이터로써 전달될 수 있다고 가정한다. 또한 ECU간 네트워크를 구축할 때 쓰이는 CAN은 데이터를 빠르면서도 고정 속도로 전송하는 것을 보장한다. 따라서 전달되는 데이터의 양이 크지 않다고 가정했을 때, 데이터 교환에 소모되는 시간을 0에 가까운 상수로 가정할 수 있다.

"시뮬레이션 정확성"은 다음과 같이 정의된다.

- 모든 시뮬레이션 태스크가 각자의 ECU에서 수행된다고 가정할 때 예상되는 수행 행태에서 도출된 모든 실세계 상호작용 시점의 데이터 교환이 동일하게 모사된다.

이러한 환경에서 우리는 시뮬레이션 태스크의 수행 행태를 H에서 정확하게 모사하고, 효율적인 시뮬레이션을 통해 시뮬레이션 정확성을 유지하면서도 모사 가능한 경우의 수를 최대화 하는 것을 목적으로 한다.

IV. 제안하는 기법

이 장에서는 시뮬레이션 호스트 H에서 시뮬레이션 정확성을 유지하면서도 시뮬레이션 태스크를 효율적으로 수행하는 기법을 본고에서 소개한다. 제안하는 접근법은 2가지 과정을 통해 이루어진다: (1) 시뮬레이션 태스크들의 작업에 따른 시간 제약 추출, (2) H에서 시간 제약에 따라 시뮬레이션 태스크들을 수행. 본고에서는 (1)에 대하여 중점적으로 설명한다.

우선 시뮬레이션 태스크들의 하이퍼 주기 (Hyper Period) 내에서 수행될 모든 작업이 실제 ECU에서 언제 시작되고 종료될지 나타내는 예상 수행 행태를 구한다. 만약 τ_1, τ_2, τ_3, τ_4 의 주기가 10이고 각 수행 시간이 ECU에서 3, 2, 2, 2라고 한다면 그림 3과 같이 보일 수 있다. 그림 3에서는 20까지만 표현되었지만 실제로는 무한한 시간에 대하여 그려진다. 결과적으로 모든 작업에 대해 구할 수 있으며 이를 표현하기 위해 다음의 기호를 정의한다:

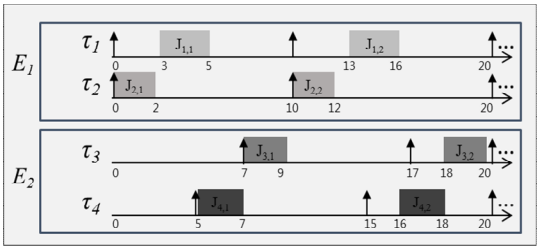


그림 3. 하이퍼 주기에 따른 예상 행태.

Fig. 3. Expected behavior by hyper period.

- $ST_e(J_{i,j})$: ECU에서 $J_{i,j}$의 예상 시작 시간
- $FT_e(J_{i,j})$: ECU에서 $J_{i,j}$의 예상 종료 시간

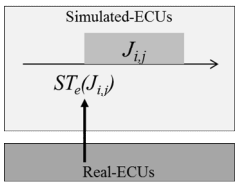
구해진 예상 수행 행태에 맞추어 H에서 각 작업을 릴리즈 타임과 데드라인에 따라 수행한다.

이때 시뮬레이션 정확성을 확보하기 위한 가장 직관적인 방법은 $ST_e(J_{i,j})$를 시뮬레이션 태스크의 릴리즈 타임과 같게 하고 $FT_e(J_{i,j})$를 데드라인과 일치시키는 것이다. 기호를 이용하여 정의하면:

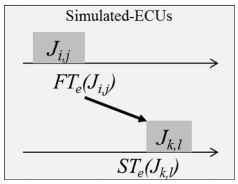
- $R_{i,j}$: 시뮬레이션 호스트에서 $J_{i,j}$의 릴리즈 타임
- $D_{i,j}$: 시뮬레이션 호스트에서 $J_{i,j}$의 데드라인

$$R_{i,j} = ST_e(J_{i,j}) \text{ and } D_{i,j} = FT_e(J_{i,j}).$$

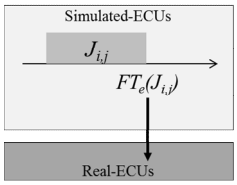
그러나 그림 2에서 보였었듯이, 시뮬레이션의 효율성을 위해서 시뮬레이션 정확성을 해치지 않는 선에서 시간 제약을 바꾸어주어야 한다. 시뮬레이션 정확성을 위한 표현을 위해 다음의 기호를 정의한다:



(a) Data reception from real-ECU.



(b) Data exchange between two simulated-ECU.



(c) Data transmission to real-ECU.

그림 4. 시뮬레이션 정확성을 위해 고려하는 세 가지 상황.

Fig. 4. Three condition for simulation correctness.

• $ST_s(J_{i,j})$: 시뮬레이션 호스트에서 $J_{i,j}$의 실제 시작 시간
• $FT_s(J_{i,j})$: 시뮬레이션 호스트에서 $J_{i,j}$의 실제 종료 시간

이제 시뮬레이션 정확성을 보장하며 시뮬레이션 행태를 모사하기 위해 고려해야 하는 상황 C1, C2, C3의 3가지를 설명한다. 그림 4는 시뮬레이션 정확성을 보장을 위해 고려해야하는 세 가지 상황을 표현하고 있다.

C1. $ST_e(J_{i,j}) \leqq ST_s(J_{i,j})$.

첫 번째는 그림 4(a)에서처럼 한 시뮬레이션 태스크의 한 작업 $J_{i,j}$가 실제 ECU로부터 데이터를 받는 경우이다. 이 경우 $J_{i,j}$는 수행 시작 전까지 실제 ECU로부터 데이터를 받을 수 있어야 하므로 H는 $ST_e(J_{i,j})$ 보다 늦게 $J_{i,j}$를 수행해야 한다.

C2. $FT_s(J_{i,j}) \leqq ST_s(J_{k,l})$.

그림 4(b)를 보자. 그림 4(b)는 시뮬레이션 태스크의 한 작업 $J_{i,j}$가 다른 ECU로 데이터를 보내는 경우이다. 이 경우 H에서는 수행 결과를 미리 가지고 있다가 해당 데이터가 쓰이는 $ST_s(J_{k,l})$ 전에만 내보내면 된다.

C3. $FT_s(J_{i,j}) \leqq FT_e(J_{k,l})$.

시뮬레이션 호스트에서 $J_{i,j}$가 $FT_s(J_{i,j})$ 지점에서 끝났을 때, 시뮬레이터는 시뮬레이터 정확성을 위해서 단지 $J_{i,j}$의 결과를 그림 4(c)처럼 $FT_e(J_{k,l})$ 시점까지 가지고 있어야 한다.

C1과 C2는 시뮬레이션 정확성의 기능적인 부분을 보장하며 C3는 시간적 부분을 보장한다. 이러한 시뮬레이션 정확성을 위한 세 가지 상황으로부터 $J_{i,j}$의 $R_{i,j}$와 $D_{i,j}$를 유도할 수 있다. 이것을 고려하기 위한 기초적인 바탕은 $J_{i,j}$의 실행시간을 가능한 한 크게 결정짓는 것이다. 다시 말해, 세 가지 조건을 어기지 않으면서 릴리즈 타임을 가능한 한 이르게 하고 데드라인을 가능한 한 늦추는 것을 말한다. $J_{i,j}$의 릴리즈 타임을 $R_{i,j}$라 하고 데드라인을 $D_{i,j}$라 하자. 이때 $R_{i,j}$와 $D_{i,j}$를 다음과 같이 정의할 수 있다:

• $pre(J_{i,j})$: $J_{i,j}$가 직접 데이터를 받아오는 전임자 집합
• $suc(J_{i,j})$: $J_{i,j}$의 결과를 직접 이용하는 후임자 집합
• $R_{i,j} = \max(R^0_{i,j}, \max_{\forall J_{g,h} \in pre(J_{i,j})}(R_{g,h} + C_g))$

여기서 $R^0_{i,j} = \begin{cases} ST_e(J_{i,j}) & J_{i,j}\text{가 실제 ECU로부터 데이터를 받아오면} \\ 0 \end{cases}$

• $D_{i,j} = \min(D^0_{i,j}, \min_{\forall J_{k,l} \in suc(J_{i,j})}(D_{k,l} - C_k))$

여기서 $D^0_{i,j} = \begin{cases} FT_e(J_{i,j}) & J_{i,j}\text{가 실제 ECU로 데이터를 내보내면} \\ \infty \end{cases}$

지금까지 H에서 시뮬레이션 정확성을 보장하면서 릴리즈 타임과 데드라인을 정하는 방법에 대하여 설명하였다. 시뮬레이션 정확성 조건을 만족시키는 $J_{i,j}$의 $R_{i,j}$과 $D_{i,j}$에 맞추어 시뮬레이션 태스크의 작업들을 스케줄링 할 때, 싱글 코어인 시뮬레이션 호스트에서 최적으로 알려진 EDF (Earliest Deadline First)를 사용함으로써 작업들을 수행시킬 수 있다. 하지만 사용자 구현 단계에서 본래의 EDF대로 선점적으로 구현하는 것은 쉽지 않아 비선점적으로 구현할 것이다. 비선점 스케줄러 중에서는 최적으로 알려진 NINP-EDF (Non-Idling and Non-Preemptive EDF)를 기본 스케줄링 알고리즘으로 고려한다[12].

추가적으로 시뮬레이션 효율성을 더욱 높이기 위해서 작업 보장을 위한 시간적 제한을 완화시키고 미리 계산된 $R_{i,j}$과 $D_{i,j}$의 예지 정보 (Clairvoyant information)를 사용할 수 있다. 그림 5는 CEDF가 NINP-EDF에 비해 어떻게 더 나은 스케줄을 제공하는지 나타낸다. 그림을 보면 두 개의 작업 $J_{1,1}$과 $J_{2,1}$이 있다. (R, C, D)로 나타날 때, R은 릴리즈 타임, C는 수행 시간, D는 데드라인을 의미한다. 현재 시간 t가 10이라고 가정할 때, NINP-EDF는 $J_{1,1}$의 릴리즈 타임이 t=10 보다 이르기 때문에 바로 수행한다. t=15에 $J_{1,1}$의 수행을 마친 후에 $J_{2,1}$를 수행하지만 데드라인을 놓친다. 반대로, CEDF는 $J_{2,1}$이 데드라인을 맞추지 못할 것을 알고 $J_{1,1}$을 t=10에 시작하지 않는다. t=11까지 지연한 이후에 J2,1을 t=14까지 수행한다. 이어서 $J_{1,1}$을 수행함으로써 $J_{1,1}$의 데드라인도 지켜진다. 이로부터 CEDF (Clairvoyant EDF)가 NINP-EDF보다 더 효율이 좋다는 것을 알 수 있다[13]. 따라서 본고에서는 시뮬레이터가 싱글 코어의 시뮬레이션 호스트에서 동작할 때, 위에서 계산된 시간 제약인 $R_{i,j}$와 $D_{i,j}$를 고려하여 시뮬레이션 태스크가 가진 작업들의 스케줄과 수행에 CEDF를 쓰는 것을 제안한다.

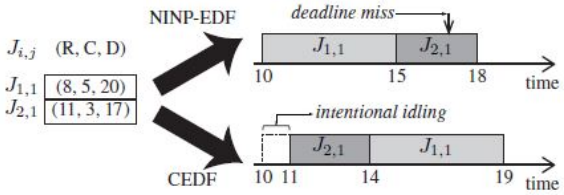


그림 5. NINP-EDF와 CEDF 간 비교.
Fig. 5. A comparsion between NINP-EDF and CEDF.

요약하면 시뮬레이션 정확성을 위한 제한 하에서 제안하고 있는 방식은 시뮬레이션 효율을 두 부분에서 극대화시킨다.

• 각 작업 $J_{i,j}$의 시간 제약 완화
• CEDF에 근거한 시뮬레이션 호스트의 의도적 지연을 이용하여 시뮬레이션 가능한 경우의 수 최대화

제안한 방식으로 인하여 시뮬레이션 능력이 얼마나 증진되었는지는 뒤의 실험 부분에서 보인다.

V. 실험 및 구현

이번 장에서는 제안된 방식을 적용시켰을 때 임의로 ECU들로 구성된 시스템에 대하여 시뮬레이션이 얼마나 확장성을 가지고 있는지 보일 것이다. 다음과 같은 매개 변수들을 가지고 다양한 대상 시스템을 구성하였다:

• 시뮬레이션 되어지는 ECU의 수는 특별한 언급이 없다면 [3,7]의 구간을 가지는 균등 분포를 따른다. 이러한 분포를 앞으로 균등 분포[3,7]과 같이 표기한다.
• 각 ECU에서 실행되는 시뮬레이션 태스크의 수는 균등 분포[1,10]을 따른다.
• 각 시뮬레이션 태스크 τ_i의 주기 P_i는 균등 분포 [10, 100] ms을 따른다.
• ECU의 태스크 τ_i의 수행 시간 C'_i는 P_i의 균등 분포 [10, 50]% 를 따른다.
• 시뮬레이션 호스트 상에서 τ_i의 수행시간 C_i는 C'_i의

30%이다.

- 모든 시뮬레이션 태스크 중에서 실제 ECU로부터 데이터를 받아오는 태스크는 별다른 언급이 없다면 30%이다.
- 모든 시뮬레이션 태스크 중에서 실제 ECU로 데이터를 내보내는 태스크는 별다른 언급이 없다면 30%이다.

시뮬레이션 ECU는 싱글 코어 시스템이고 RM (Rate Monotonic) 스케줄링 정책을 이용한다고 가정한다. 또한 시뮬레이션 태스크는 항상 스케줄 가능하다고 가정한다. 태스크 간 데이터 의존성은 무작위하게 설정된다. 본고에서는 시뮬레이션 가능 수를 다음 네 가지의 시뮬레이션 정확성을 보장하는 시뮬레이션 접근법에 대하여 측정하였다:

- strict + no-idle(엄격한 제약 및 지연 제외). 시뮬레이션 태스크의 작업을 예상 시작 시간에 맞추어 릴리즈하고, NINP-EDF가 아닌 비선점 EDF 스케줄링 알고리즘을 이용한 기본적인 접근 방식이다.
- strict + idle(엄격한 제약 및 지연 포함). 시뮬레이션 태스크의 작업을 NINP-EDF를 가지고 수행하는 것 이외에는 “strict + no-idle”과 동일하다. 따라서 이 접근법을 통해 지연 (idling)이 포함될 때 시뮬레이션 가능 수가 증가한다는 것을 알 수 있다.
- relaxed + no-idle(완화된 제약 및 지연 제외). “strict + no-idle(엄격한 기준 및 지연 제외)”에서 시간 제약이 완화된 $R_{i,j}$들과 $D_{i,j}$들을 사용하지 않을 때의 양상을 보여준다. 그래서 이 접근법은 “완화된 시간 제약”을 사용함으로써 시뮬레이션 가능 수를 증가시킬 수 있다는 것을 보여준다.
- relaxed + idle(완화된 제약 및 지연 포함). “의도적 지연”과 “완화된 시간 제약”이 모두 포함된, 최종적으로 제안한 접근 방식이다.

그림 6에서 시뮬레이션 호스트가 싱글 코어 시스템일 때, 접근 방식의 차이에 따른 시뮬레이션 가능 수를 보였다. 이 그림을 구성하기 위해서 앞서 언급했던 매개 변수들에 따라 500개의 대상 시스템을 구성했고 이는 Y축에 나타난다. 하이퍼 주기를 고려하여 가상으로 시뮬레이션 호스트의 스케줄링 다이어그램을 그림으로써 실제 코드를 수행하지 않고도 특정 대상 시스템이 시뮬레이션 가능한지 아닌지를 확인할 수 있었다. 결과적으로 본고에서 제안한 방식 중 하나를 적용했을 때 두 방식 모두 기본적인 접근 방식에 비해 약 7배 증가한 것을 확인했다. 두 가지의 개념을 모두 적용했을 땐 15배까지 증가했다.

그림 7과 그림 8에서는, 시스템이 ECU를 3개부터 7개까지 포함할 때 시뮬레이션 가능 수 및 시뮬레이션 가능 여부 측정 시간을 표현했다. 제안하고 있는 접근 방식인 CEDF를 사용하는 “relaxed + idle”과 모든 경우의 수를 총망라하는 방식인 Exhaustive search를 쓴 “relaxed + optimal”를 비교했다.

그림 7에서 “relaxed + optimal” 방식은 시뮬레이션 작업이 실행되기 전에 미리 스케줄링 다이어그램을 전부 그려야하기 때문에, 하나의 하이퍼 주기에 대해서만 스케줄링 다이어그램을 그려보면 되는 “relaxed + idle”과 비교하여

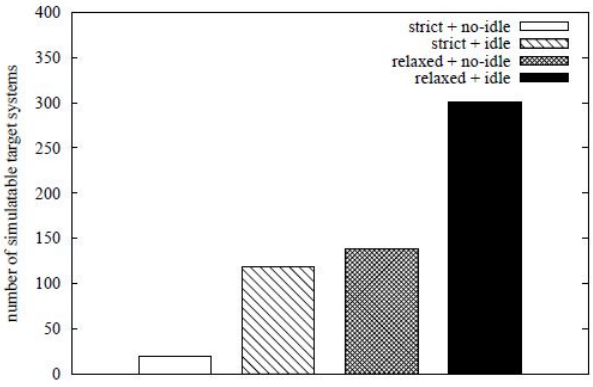


그림 6. 싱글 코어 시스템에서 시뮬레이션 가능 수.

Fig. 6. Simulation capacity on the singlecore system.

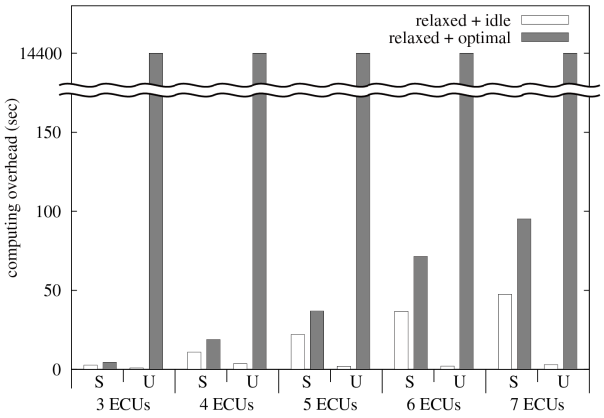


그림 7. 싱글 코어 시스템에서 ECU의 수에 따른 시뮬레이션 가능 여부 측정 시간.

Fig. 7. Simulation capacity and the time for checking the simulatability on the singlecore system as scaling up the number of ECUs in the target system.

명확하게 연산량이 많아 “relaxed + optimal”이 더 오래 걸릴 것이라고 실험 전에도 간단히 추측할 수 있을 것이다. 최적의 스케줄을 찾을 때, 4시간동안 시뮬레이션 가능한 경우를 못 찾으면 시간을 더 써서 찾더라도 적용이 힘들기 때문에 실패한 것으로 간주하였다. 그림에서는 값의 차이가 커서 Y축을 무의미하게 만들어 버리므로 물결 표시를 이용하여 데이터의 간격이 크다는 것을 표현했다. X축의 경우 대상 시스템을 구성한 ECU의 수 이외에도 S (Simulatable), U (Unsimulatable)로 시뮬레이션 가능 여부에 따라 스케줄링 다이어그램을 찾는 시간을 추가적으로 나타냈다.

그림 8에서는 시스템을 구성하는 ECU의 수에 따른 시뮬레이션 가능 수를 보인다. “relaxed + optimal”이 “relaxed + idle”보다 조금 더 찾아내고 있다. 탐색 시간을 고려하지 않고 모든 경우의 수에 대하여 확인하기 때문에 “relaxed + idle”로는 찾지 못한 경우에 대해서도 시뮬레이션 가능한 스케줄을 발견하기도 한다.

따라서 “relaxed + idle”은 시뮬레이션 가능 여부를 “relaxed + optimal”보다 2배 정도 빠르게 찾을 수 있음을 보였다. 반면에, 시스템이 시뮬레이션 가능하지 않을 경우

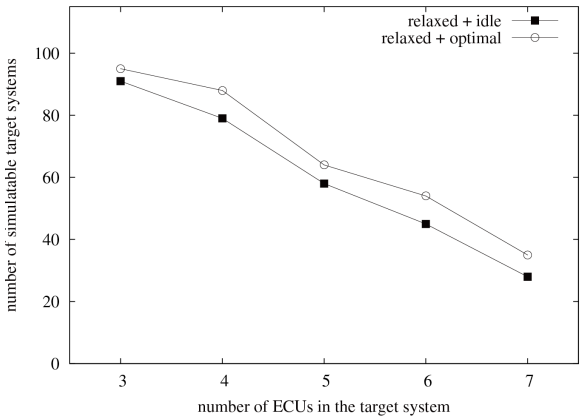


그림 8. 싱글 코어 시스템에서 ECU의 수에 따른 시뮬레이션 가능 수.

Fig. 8. Simulation capacity on the singlecore system as scaling up the number of ECUs in the target system.

"relaxed + idle"의 연산량 오버헤드는 "relaxed + optimal"에 비해 훨씬 적었다. 뿐만 아니라 모든 시뮬레이션 불가능한 경우에 대해서 4시간을 모두 소모함으로써 "relaxed + optimal"은 주어진 시뮬레이션 태스크들의 작업에 대하여 모든 경우의 수를 살펴보는 것이 어렵다고 보였다. 하지만 본고에서 제안하는 "relaxed + idle"은 시뮬레이션이 불가능하다는 것을 평균 10초 내로 보여 다른 방식과 비교해 보았을 때 쓰이기에 충분히 허용 가능함을 보였다.

마지막으로, 우리는 제안한 시뮬레이터의 정확성을 보이기 위해서 실시간으로 차량 동력학을 시뮬레이션하고 차량의 물리적 행태를 3D 애니메이션으로 시각화해주는 차량동역학 시뮬레이터 CarSim [14]을 사용하였다. 시뮬레이션 호스트와 TriCore TC1797 마이크로프로세서[15] 위에 실제로 구현된 ECU는 CAN 버스를 통해 CarSim에 연결시켰다. 3대의 ECU {ECU_1, ECU_2, ECU_3} 로 구성된 정속 주행 및 차선 유지 시스템을 대상으로 하였다. ECU_1은 차량의 현재 속도와 가속도를 감지하여 정속 주행하는 기능을 한다. ECU_2는 차량이 주행하고 있는 선로에서, 상대적으로 차량의 측면이 어디에 위치해 있는지 파악한다. ECU_3는 차선을 유지하기 위해 적당한 차량의 조향각을 계산하고 액추에이터에 결과를 내보낸다.

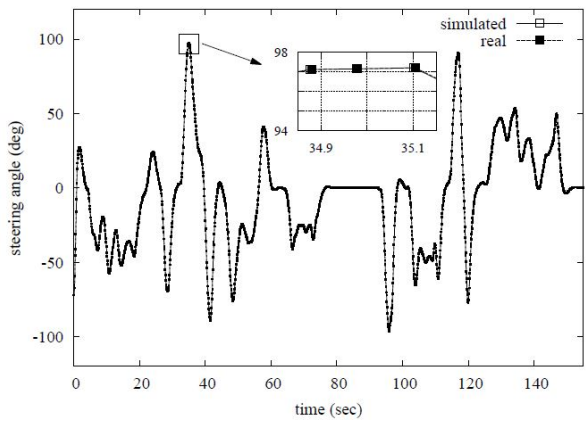


그림 9. 실제 ECU와 시뮬레이터의 결과 비교.

Fig. 9. Result compparison between the real ECU and the simulator.

이러한 시스템을 대상으로 하여 3대의 ECU를 실제의 ECU로 구성하였을 때와 ECU_3만 시뮬레이터가 대체하였을 때를 비교하였다. 그림 9는 시뮬레이터 ECU_3와 실제 ECU_3에서 내보내진 결과 값을 그린 그래프이다. X축은 차량이 주행을 시작하고 나서부터의 시간이며 Y축은 CarSim의 차량에 적용될 조향각이다. 그림 9에서 보이고 있듯이 그래프의 두 곡선은 거의 정확히 일치하는 것을 볼 수 있다.

VI. 결론 및 향후 연구

본고에서는 자동차 제어 시스템의 정확하고 효율적인 시뮬레이션을 위한 기법이 제안되었다. 본고에서 제안된 기법은 기존의 기법보다 더 많은 경우의 수에 대해서도 안전하게 시뮬레이션 가능하며 시뮬레이션 호스트와 ECU의 차이를 인식함으로써 보다 현실적이라는 강점을 가진다.

추후에는 멀티코어 (Multi-Core)를 지원하는 ECU를 멀티코어 시뮬레이션 호스트를 가지고 효율적으로 시뮬레이션을 구성하는 방안을 검토하고자 한다. 또한 본고에서 제안하는 시뮬레이션을 이용하여 복잡한 시스템을 설계할 때 발생할 수 있는 문제 및 해결 방법을 실제 예를 통해 보이고자 한다[16].

REFERENCES

[1] D. W. Kang, S. Lee, and G. C. Lee, "Formation performance evaluation of warm robot control method using simulation," *Institute of Control, Robotics and Systems (in Korean)*, vol.20, no.4, pp. 23-24, May 2014.

[2] M. Broy, "Challenges in automotive software engineering," *International Conference on Software Engineering (ICSE)*, pp. 33-42, 2006.

[3] Mathworks, "Matlab/Simulink," http://www.mathworks.co.kr/products/simulink/.

[4] dSPACE, "SCALEXIO," http://www.dspace.com/en/pub/home/products/hw/simulator_hardware_scalexio.cfm/.

[5] F. Bellard. "QEMU, a fast and portable dynamic translator," *Usenix Annual Technical Conference (ATEC)*, pp. 41-46, Apr. 2005.

[6] Imperas Software, "Open virtual platforms," http://www.ovpworld.org.

[7] D. Mueller-Gritschneder, K. Lu, and U. Schlichtmann, "Control-flow-driven source level timing annotation for embedded software models on transaction level," *Euromicro Conference on Digital System Design (DSD)*, pp. 600-607, Aug. 2011.

[8] S. Roloff, F. Hannig, and J. Teich, "Fast architecture evaluation of heterogeneous MPSoCs by host-compiled simulation," *International Workshop on Software and Compilers for Embedded Systems (SCOPES)*, pp. 52-61, May 2012.

[9] P. Razaghi and A. Gerstlauer, "Host-compiled multicore RTOS simulator for embedded real-time software development," *Automation and Test in Europe Conference (DATE)*, pp. 1-6, Mar. 2011.

[10] C. Dufour, C. Andreade, and J. Belaneger, "Real-time simulation technologies in education: a link to modern engineering methods and practices," *International Conference on Engineering and Technology Education (INTERTECH)*, Mar. 2010.

[11] dSPACE GmbH, "AutoBox," https://www.dsapce.com/en/ltd/home/products/hw/accessories/autobox.cfm.

[12] K. Jeffay, R. Anderson, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," *IEEE Real-Time Systems Symposium (RTSS)*, pp. 129-139, Dec. 1991

[13] C. Ekelin. "Clairvoyant non-preemptive EDF scheduling," *Euromicro conference on Real-Time Systems (ECRTS)*, pp. 23-32, Jul. 2006.

[14] Mechanical Simulation Corporation, *CarSim*. http://www.carsim.com/.

[15] Infineon Technologies AG, *TC1797*. http://www.infineon.com/cms/en/product/channel.html?channel=db3a30431f848401011fa273b2eb3473.

[16] D. G. Oh, H. M. Lee, and M. C. Lee, "Development of unmaned driving system of mini electronic vehicle," *Institute of Control, Robotics and Systems (in Korean)*, vol. 20, no. 4, pp. 498-499, May 2014.

김 승 곤

2013년 광운대학교 컴퓨터공학과(공학사). 2013년~현재 서울대학교 대학원 컴퓨터공학부 석박통합과정 재학중. 관심분야는 실시간 임베디드 시스템, CPS 시뮬레이션, 무인차 소프트웨어 플랫폼 등.

위 경 수

2009년 서울대학교 컴퓨터공학부(공학사). 2011년 서울대학교 컴퓨터공학부(공학석사). 2011년~현재 서울대학교 대학원 박사과정 재학중. 관심분야는 실시간 임베디드 시스템, 플래시 저장장치 시스템, 자동차 소프트웨어 등.

이 창 건

1991년 서울대학교 컴퓨터공학부(공학사). 1993년 서울대학교 컴퓨터공학부(공학석사). 1998년 서울대학교 컴퓨터공학부(공학박사). 2006년~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 실시간 임베디드 시스템, 사이버 물리 시스템, 유비쿼터스 시스템 등.

이 경 수

1985년 서울대학교 기계공학과(공학사). 1987년 서울대학교 기계공학과(공학석사). 1992년 University of California. Mechanical engineering (공학박사). 현재 서울대학교 기계항공공학부 교수. 관심분야는 제어 시스템, 운전자 보조 시스템, 지상차 능동적 안전 시스템 등.