Journal of Institute of Control, Robotics and Systems (2013) 19(3):248-255
http://dx.doi.org/10.5302/J.ICROS.2013.12.1823 ISSN:1976-5622 eISSN:2233-4335

OPRoS 기반 로봇시스템의 Robot-in-the-Loop Simulation 구조

Design of a Robot-in-the-Loop Simulation Based on OPRoS

김 성 훈, 박 홍 성*
(Seong-Hoon Kim[1] and Hong Seong Park[1])
[1]Kangwon National University

Abstract: This paper proposes the architecture of the RILS (Robot-in-the-Loop-Simulation) consisting of the robot, the virtual robot, and the avatar robot which is the type of virtual robots operating according to the robot status and behavior. And the synchronization algorithm for mobilization part of the avatar robot is suggested, which reduces the difference between behaviors of the robot and those of the avatar robot. This difference occurs due to the environmental and mechanical mismatches between the robot and avatar robot. In order to reduce this difference in robots behaviors, the synchronization algorithm controls the avatar robot based on the data observed from the robot's behavior. The proposed architecture and the synchronization algorithm are validated from some simulation results.

Keywords: OPRoS, simulation, HILS, RILS, HIL driver, synchronization

I. 서론

최근 로봇시스템은 로봇 제어 S/W가 컴포넌트 혹은 S/W 모듈로 개발되는 추세이며, 이를 이용하여 로봇응용을 빠르게 개발함으로써 개발 효율을 증대시킨다. 이러한 로봇 시스템은 개발 초기단계부터 동작에 대한 검증이 필요하며 가장 널리 이용하고 있는 검증 방법에는 컴퓨터 시뮬레이션을 사용하는 방법이 있다. 시뮬레이션을 이용한 로봇의 모의실험은 적은 비용과 짧은 시간 안에 큰 효과를 볼 수 있다는 장점이 있어 널리 사용되고 있지만, 모든 H/W적 특성이 수식적 모델로 대체되어 실험되므로 실제 로봇시스템에 이식되었을 시에 동일한 동작을 보장할 수 없다. 따라서 로봇 시스템의 효과적인 검증을 위해서는 시뮬레이션의 장점과 실제 로봇의 H/W 및 S/W적 특성을 포함하는 새로운 시뮬레이션 기법이 요구되며, 이를 통해 실제 로봇의 특성을 기반으로 시뮬레이션에서 제공하는 다양한 환경에서 로봇을 검증할 수 있어야 한다. 일반적으로 H/W 일부를 시뮬레이션으로 대체하는 것을 HILS (Hardware-in-the-Loop Simulation)라 하고, 실제 로봇 시스템과 시뮬레이션이 같이 연동하여 동작하는 것을 RILS (Robot-in-the-loop Simulation)라 한다.

RILS는 하나 이상의 실제 로봇 시스템들이 시뮬레이션 상에서 가상로봇과 동시에 동작하는 시뮬레이션 기법을 말한다. 기본적으로 RILS는 HILS (Hardware-in-the-Loop-simulation)를 기반으로 하지만 대상이 H/W 부품단위가 아니라 로봇 시스템이라는 차이가 있다. HILS란 실제 실험을 통해 검증하기 힘든 위험기반 환경을 시뮬레이션으로 제작하여 H/W의 동작을 검증하기 위한 시뮬레이션 기법[2]이며 현재까지 차량[3-5], 철도[6], 항공[7], 제어[8-10], 산업용 머니퓰레이터[11], 소형무인기 군집비행 알고리즘[16] 등의 시험에서 성공적으로 적용되어 왔다. 기존 HILS에 대한 연구[3-11,16]에서는 주로 전체 시스템 중 제어를 담당하는 ECU 혹은 임베디드 보드와 시뮬레이션 모델이 루프로 구성되므로 전자장치로 구성된 제어 H/W를 사용하여 시뮬레이션 환경에서도 실제 시스템과 유사하게 검증할 수 있다는 특징이 있다. 여기서 루프를 구성한다는 말은 시스템에서 중요한 역할의 H/W와 시뮬레이션이 연결됨을 말한다. 또한 이러한 검증 시스템은 실제 개발된 소스 코드를 기반으로 하기 때문에 SW 공학도구[17]와 같이 활용하면 매우 체계적으로 검증할 수 있다.

RILS는 기본적으로 HILS의 장점을 수용하지만 좀더 복잡하고 큰 규모의 로봇 시스템의 검증에 유용하다[12]. 예를 들어 로봇의 동작을 검증하기 위한 환경구성이 어려운 경우, RILS를 통해 실제 로봇은 시뮬레이션의 가상 환경에서 동작할 수 있으며, 복수의 로봇이 동시에 동작하는 멀티로봇 시스템에서 소수의 실제 로봇과 다수의 가상로봇(virtual robot)이 시뮬레이션 환경에서 동시에 동작할 수 있다.

RILS에 대한 연구[12]는 최근에서야 진행되고 있어 거의 연구가 없으며 주로 제어 H/W가 시뮬레이션과 루프로 연결되어 동작하는 HILS를 확장하여, 로봇시스템 자체와 시뮬레이션이 루프로 연결되는 RILS 개념을 도입하였다. 이를 위해 [12]에서는 인터페이스 정의하여 실제 로봇의 동작을 시뮬레이션 환경의 가상로봇에 재현하였으며, 다른 가상로봇들과 동시에 동작하게 함으로써 RILS를 통해 로봇을 검증하였다. 본 논문에서는 이러한 로봇을 구분하기 위하여 전자를 실제로봇이라 하고 후자를 아바타 로봇이라 하며 순수 시뮬레이션만으로 동작하는 로봇모델을 가상로봇이라고 한다. 하지만 RILS의 동작 시에 실제 환경에서 동작하는 실제로봇과 시뮬레이션에서 동작하는 아바타 로봇은 각각 실제/가상의 환경변수에 영향을 받게 되므로 동일한 명령을 수행한다고 하여도 환경변수 차이에 따라 결과가 동일하다고 보장할 수 없었다. 따라서 이러한 차이는 중첩될수록 실제로봇과 아바타로

* 책임저자(Corresponding Author)
논문접수: 2012. 9. 11., 수정: 2012. 11. 2., 채택확정: 2013. 1. 4.
김성훈, 박홍성: 강원대학교 전자통신공학과
(lemonade702@control.kangwon.ac.kr/hspark@control.kangwon.ac.kr)
※ 이 논문은 교육과학기술부 및 대구경북과학기술원의 지원을 받아 수행하였음.

Copyright© ICROS 2013

봇이 서로 다르게 동작할 수 있다는 한계가 있으므로 아바타 로봇과 실제로봇간 동작차이를 보정할 수 있어야 한다.

본 논문에서는 로봇의 이동부분에 한하여 실제로봇과 아바타로봇의 이동 차이를 보정하기 위한 아바타 로봇용 RILS 구조를 제안한다. 제안하는 아바타 로봇용 RILS 구조에서는 "HIL 드라이버"를 생성하여 실제 로봇과 아바타 로봇에서 발생할 수 있는 차이를 지속적으로 감지하며, 실제로봇 H/W 정보를 기반으로 아바타로봇의 동작을 보정한다. H/W관련 컴포넌트의 경우 H/W 제어를 위한 디바이스 드라이버 대신 모듈화된 HIL 드라이버를 사용하는 구조를 통해 로봇 H/W와 시뮬레이션이 연결된다. 특히 로봇이 동작하는 환경상의 이유로 발생할 수 있는 실제 로봇과 아바타 로봇의 동작차이를 감소시키기 위해, HIL 드라이버의 동기화 모듈에서는 실제로봇 H/W 정보를 기반으로 아바타 로봇의 H/W 동작을 보정한다. 또한 RILS 수행 시에 HIL 드라이버의 사용만으로 로봇 컴포넌트나 시뮬레이션의 변경 없이 RILS가 동작할 수 있는 효과를 보여준다. 본 논문에서 제안하는 방법의 검증을 위해 컴포넌트 기반의 OPRoS [1] 기반 로봇시스템에 구현하고, 다양한 속도 프로파일을 활용하여 RILS 적용 전후의 로봇 이동 거리의 오차를 비교하여 유효성을 검증한다.

본 논문은 다음과 같이 구성된다. 제 II 장에서는 OPRoS 컴포넌트 동작에 대한 전반적인 설명을 기술하고 제 III 장에서는 OPRoS RILS의 구조에 대해 설명하며 제 IV 장에서는 컴포넌트 기반의 OPRoS를 이용하여 제안하는 RILS의 구현 사항 및 결과에 대해 기술한다. 마지막으로 V 장에서는 결론을 맺도록 한다.

II. OPRoS 컴포넌트 동작 개요

OPRoS는 로봇 소프트웨어가 컴포넌트로 개발되며 OPRoS 컴포넌트에서는 외부 컴포넌트와의 인터페이스가 포트를 통해 이루어진다. OPRoS 컴포넌트 포트의 종류는 서비스 포트, 데이터 포트, 이벤트 포트로 구분되며 컴포넌트의 포트연결을 통해 로봇을 제어할 수 있는 로봇응용을 제작할 수 있다. 그림 1의 예는 LaserScanner 컴포넌트, WheelController 컴포넌트, 그리고 제어를 위한 RobotControl 컴포넌트의 서비스 및 데이터 포트연결을 통한 로봇응용의 구성을 보여준다. 해당 로봇응용은 RobotControl 컴포넌트와 WheelController 컴포넌트가 서비스 포트로 연결되고, RobotControl 컴포넌트와

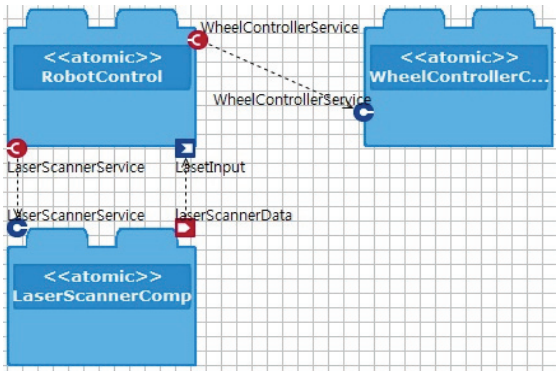

그림 1. OPRoS 컴포넌트 구성의 예.
Fig. 1. Example of OPRoS component composition.

```
<properties>
<!--컴포넌트에서 사용한 디바이스 드라이버 파일의 정보-->
<property name="APIName" type="string">KNU_HRC01_MC_V20.dll
</property>
<!--장치 축 지름-->
<property name="WheelDiameter" type="double">0.13</property>
<!--장치 축 길이-->
<property name="AxleDistance" type="double">0.30</property>
<!--장치 분산 길이-->
<property name="VarianceDistance" type="double">0.05</property>
<!--장치 분산 방향-->
<property name="VarianceDirection" type="double">1.0</property>
<!--장치 반지름-->
<property name="SafeRadius" type="double">0.3</property>
<!--장치 최고속도-->
<property name="MaximumVelocity" type="double">0.2</property>
<!--장치 가속도-->
<property name="Acceleration" type="double">0.5</property>
</properties>
```

그림 2. WheelController 컴포넌트 프로파일의 예.
Fig. 2. Example of wheelcontroller component profile.

LaserScanner 컴포넌트가 서비스, 데이터 포트로 연결된다. 이러한 포트연결을 통해 해당 로봇은 레이저 스캐너 센서의 장애물 측정정보를 이용하여 이동로봇을 주행시키는 동작을 수행한다.

OPRoS 컴포넌트는 확장자가 *.dll(윈도우즈) 혹은 *.so(리눅스)인 바이너리 파일과 수행에 필요한 속성정보가 xml로 저장되는 컴포넌트 프로파일로 구성된다. 그 중 컴포넌트 프로파일의 경우에는 컴포넌트의 개략적인 설명, 수행에 필요한 정보, 그리고 포트정보가 기록되며 속성정보의 예는 그림 2와 같다. 그림은 로봇의 바퀴를 제어하는 WheelController 컴포넌트 프로파일 속성정보의 내용이며 H/W 제어를 위한 디바이스 드라이버(*.dll)의 정보와 바퀴제어에 필요한 바퀴 지름, 길이, 방향 등의 정보로 구성된다.

OPRoS 컴포넌트는 순수 알고리즘 S/W로 이루어진 컴포넌트와 H/W와 연관된 컴포넌트로 나눌 수 있으며, 그 중 WheelController 컴포넌트와 같이 H/W 연관된 컴포넌트의 경우에는 기본적으로 H/W 추상화 모델을 사용하는 OPRoS 디바이스 드라이버 정보를 컴포넌트 프로파일에 입력시키면, 그림 3과 같이 H/W 연관된 컴포넌트에서 디바이스 드라이버를 적재하며, 컴포넌트의 호출에 따라 실제 H/W를 제어함으로써 실제환경에서 로봇을 동작시킨다.

특히 OPRoS 디바이스 드라이버는 실제 디바이스 드라이버에서 제공하는 API를 캡슐화 시키는 역할을 함으로써 H/W 드라이버의 변경 시에도 별도의 컴포넌트를 변경하지

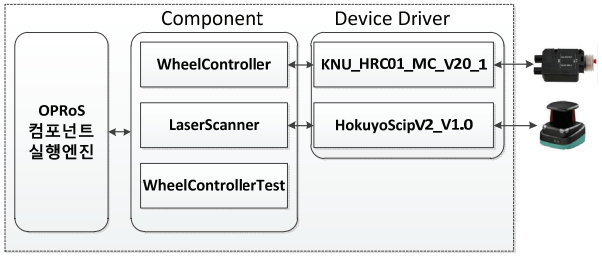

그림 3. 실제로봇 구조.
Fig. 3. Structure of robot.

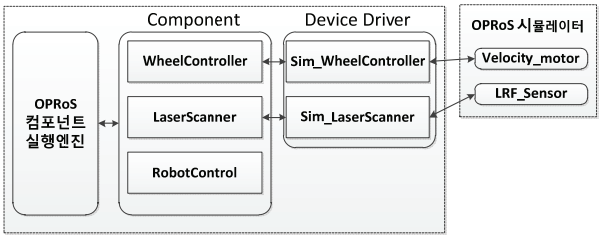

그림 4. 가상로봇 구조.
Fig. 4. Structure of virtual robot.

않고 사용할 수 있도록 도와준다. 다시 말해 디바이스 드라이버의 변경 시 OPRoS 컴포넌트 프로파일의 디바이스 드라이버 속성정보를 변경함으로써 다양한 H/W를 지원할 수 있다. 이와 달리 시뮬레이션을 이용하여 가상로봇을 동작시키는 경우에는 컴포넌트 프로파일의 속성정보에 시뮬레이션 가상 H/W의 제어기능을 갖는 디바이스 드라이버와 가상로봇 모델의 정보 및 시뮬레이션 포트 정보를 함께 입력하면 그림 4와 같이 H/W 연관된 컴포넌트에서는 시뮬레이션 용도의 가상 디바이스 드라이버를 적재한다. 로봇의 동작 시에는 컴포넌트가 시뮬레이션의 가상 H/W를 제어함으로써 시뮬레이션 환경에서 가상로봇을 동작시킬 수 있다.

이와 같이 H/W와 연관된 OPRoS 컴포넌트는 추상화된 인터페이스를 통해 로봇의 H/W를 제어함으로써 컴포넌트의 구조는 항상 동일하며, 실제 H/W 혹은 시뮬레이션의 가상 H/W를 제어하는 디바이스 드라이버를 사용함으로써 실제로봇을 동작시키거나 시뮬레이션 환경의 가상로봇을 동작시킬 수 있다.

III. OPRoS RILS 구조

RILS는 실제로봇 자체의 동작이 시뮬레이션 루프에 포함되어 시뮬레이션의 가상환경에서 동작하는 특징을 지닌다. 따라서 실제로봇의 동작이 시뮬레이션 환경에서 재현되어야 하며 이를 위해 모델링된 로봇모델을 아바타로봇이라 한다. 아바타로봇은 실제로봇과 일대일 대응되는 로봇모델로써 실제로봇의 외형적 모습과 장착된 H/W 종류 및 위치까지 동일하게 모델링 되고 실제로봇의 동작을 시뮬레이션의 가상환경에서 재현한다. 이에 따라 아바타로봇은 실제로봇의 물리적 특성을 이용하여 시뮬레이션의 환경 및 가상로봇과 동시에 동작할 수 있다. 즉 실제로봇과 아바타로봇의 동작이 유사하다면 실제로봇의 물리적 특성을 이용하여 시뮬레이션의 가상환경에서 동작을 검증할 수 있다. 실제로봇과 아바타로봇의 동작을 일치화하기 위해서는 시뮬레이션 자체를 제어하는 방법이 있지만 H/W의 추가에 따라 시뮬레이션도 지속적으로 변경되어야 하는 비효율적인 구조이므로 본 논문에서는 시뮬레이션을 변경하지 않도록 하였다.

RILS에서 로봇시스템은 실제환경과 가상환경에서 동시에 동작한다. 따라서 실제로봇과 아바타로봇은 자신이 위치한 환경의 중력, 마찰 등의 변수와 실제/가상의 H/W의 특성에 영향을 받아 동작하므로 동일한 명령에도 실제로봇과 아바타로봇이 다르게 동작할 수 있다.

실제로봇과 시뮬레이션의 가상로봇이 그림 5 a)의 속도 프로파일를 이용하여 이동하는 명령을 수행한 결과는 그림 b)

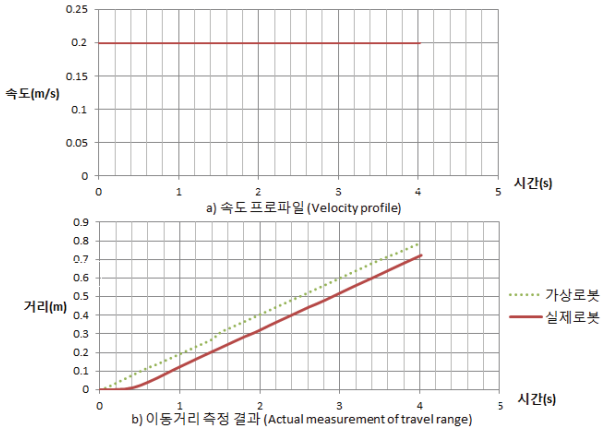

그림 5. 환경차이로 인한 동작의 차이.
Fig. 5. Difference in behavior due to environmental differences.

와 같다. 그림 5에서 실제로봇이 동작하는 경우에는 동작 초기 시점에 H/W 구동에 관련된 시간이 소요되는 반면, 가상로봇의 경우에는 초기 구동시간이 실제로봇과 비교하여 덜 소요되는 것을 확인할 수 있다. 동작이 시작되면 실제로봇과 가상로봇은 자신이 위치한 환경과 H/W 특성에 영향을 받아 동작하므로, 결과적으로 동일한 명령을 수행하는 실제로봇과 가상로봇의 동작에는 차이가 발생한다. 실제로봇의 동작을 표현하기 위한 아바타로봇 역시 가상로봇의 일종이므로 동일 명령 수행 시에 실제로봇과 동작의 차이가 발생할 수 있다. 따라서 실제로봇과 아바타로봇의 동작 차이를 보정하기 위한 기능이 요구되며 본 논문에서는 이런 기능을 동기화라고 한다. 동기화는 실제로봇의 H/W 동작 정보를 기반으로 아바타로봇의 동작을 보정하며, 결과적으로 환경과 H/W 특성에 따른 실제로봇의 동작이 아바타로봇에서 유사하게 표현된다.

기본적으로 실제로봇과 시뮬레이션의 동작에는 상호 연관성이 없지만 RILS는 실제로봇과 시뮬레이션의 아바타로봇이 연관되어 동시에 동작하므로 실제로봇의 H/W 시스템과 시뮬레이션이 서로 상호 데이터를 교환할 수 있어야 한다. 여기서 OPRoS 컴포넌트 모델은 II 장에서 언급했듯이 추상화된 디바이스 드라이버를 통해 실제 혹은 가상의 H/W를 제어할 수 있으므로, OPRoS RILS에서는 실제와 가상의 H/W에 모두 접근 및 처리하기 위한 새로운 디바이스 드라이버가 사용되며 이것을 HIL (Hardware-in-the-Loop) 드라이버라고 한다. 그림 6은 실제로봇과 아바타로봇의 주행을 동기화 하기 위

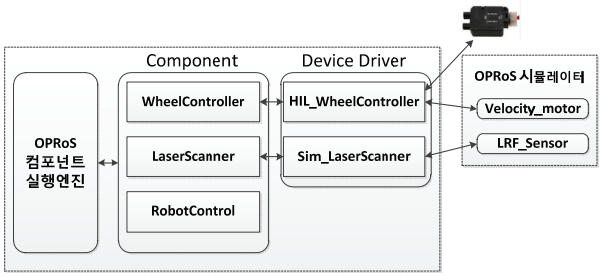

그림 6. OPRoS RILS 수행 구조.
Fig. 6. Structure of OPRoS.

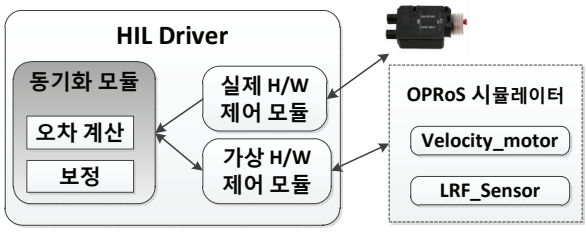

그림 7. HIL 드라이버 구성.
Fig. 7. Composition of HIL driver.

해, 로봇의 이동에 관련된 WheelController 컴포넌트의 디바이스 드라이버로 HIL 드라이버가 사용된 예시이며 기존 디바이스 드라이버 위치에 HIL 드라이버가 위치함을 보여준다.

HIL 드라이버는 그림 7과 같이 실제로봇의 제어를 위한 실제 H/W 제어모듈, 아바타로봇의 제어를 위한 가상 H/W 제어모듈, 그리고 실제로봇과 아바타로봇의 동작차이를 보정하기 위한 동기화 모듈로 구성된다.

첫 번째로 실제 H/W 제어 모듈은 HIL 드라이버에서 실제 H/W와 연관된 모든 기능을 포함한다. OPRoS 컴포넌트는 HIL 드라이버의 실제 H/W 제어 모듈을 이용하여 H/W의 상태를 초기화, 활성화, 비활성화 시키며 H/W를 제어한다. HIL 드라이버는 기본적으로 제어 대상의 H/W 디바이스 드라이버와 동일한 추상 인터페이스로 구성된다. 따라서 OPRoS 컴포넌트는 HIL 드라이버를 디바이스 드라이버와 동일하게 인식하게 되며, 결과적으로 컴포넌트는 디바이스 드라이버를 사용하는 방식과 동일하게 실제로봇의 H/W를 제어할 수 있다. 또한 실제 H/W 제어모듈은 동작중인 H/W의 수치화된 정보를 동기화 모듈에 지속적으로 전송하여 실제로봇과 아바타로봇의 차이를 보정하기 위한 기본정보를 제공한다.

두 번째로 가상 H/W 제어 모듈은 시뮬레이션과 통신하여 아바타로봇을 제어하기 위한 역할을 수행한다. 가상 H/W 제어 모듈은 시뮬레이션과의 통신기능을 통해 시뮬레이션에 등록된 아바타로봇과 연결된다. 일단 가상 H/W 제어모듈이 아바타로봇과 연결된 이후에는 아바타로봇을 구성하는 관절 및 기타 가상 H/W에 대한 목록을 구성할 수 있으며 아바타로봇의 가상 H/W를 제어할 수 있다. 또한 가상 H/W 제어모듈 역시 실제 H/W 제어모듈과 마찬가지로 가상환경에서 동작중인 아바타로봇 H/W의 수치화된 정보를 동기화 모듈에 지속적으로 전송하여, 실제로봇과 아바타로봇의 차이를 보정하기 위한 정보를 제공한다.

마지막으로 동기화 모듈은 실제로봇과 아바타 로봇의 환경상의 차이와 기계적 특성에 따라 발생할 수 있는 동작 차이를, 아바타 로봇이 실제로봇과 근접하게 동작하도록 보정한다. 동작중인 실제로봇과 아바타 로봇의 보정을 위해서는 하나의 기준이 요구되며 본 논문에서는 실제 로봇 H/W의 정보를 기준으로 아바타 로봇의 동작을 보정했다. 아바타 로봇 동작의 보정을 위해, 동기화 모듈은 실제 H/W 제어모듈과 가상 H/W 제어모듈에 실제로봇의 H/W 정보와 아바타 로봇의 H/W 정보를 지속적으로 요청한다. 특히 로봇의 H/W는 동작에 따라 활성상태와 유휴상태로 구분될 수 있지만 동기화 모듈에서는 로봇의 상태와 관계없이 지속적으로 실제로봇의 H/W 정보와 아바타 로봇의 H/W 정보를 획득할 수 있도록 했다. 예를 들어 로봇 주행의 경우, 로봇은 제어명령에 따라 도착완료 후에 유휴상태가 될 수 있지만 로봇의 가속도에 따라 유휴상태 이후에도 로봇이 이동할 수 있으므로 로봇의 모든 상태에 대해 지속적으로 H/W 정보를 획득해야 한다. 또한 동기화 모듈에서 관심의 대상은 OPRoS 컴포넌트에서 발생하는 일련의 실행 흐름이 아닌 현재 H/W의 정보이다. 다시 말해 H/W와 연관된 컴포넌트라 하여도 H/W 제어와 관련 없는 작업을 수행할 수 있기 때문에 컴포넌트의 실행흐름에 따라 동기화를 수행하는 것은 무의미하며, 만약 H/W 제어가 블로킹 상태로 수행될 경우 실제로봇과 아바타로봇의 동기화에는 시간지연이 증가할 수 있다. 동기화 모듈은 실제 H/W 제어모듈과 가상 H/W 제어모듈로부터 획득한 실제 및 가상 H/W 정보를 기반으로 실제로봇과 아바타로봇의 동작 차이를 계산하며, 위치보정 작업에서는 가상 H/W 제어모듈을 이용하여 오차계산결과에 따라 아바타로봇을 제어하여 동작을 보정한다.

IV. 구현 및 결과

제안한 RILS의 유효성을 검증하기 위하여 실제 H/W모델 및 그와 관련된 시뮬레이션 아바타로봇을 제작하였다. 표 1에는 각 로봇에 사용된 모듈과 설정변수가 나열되어 있으며 그림 8에 외형에 제시되어 있다.

실제로봇은 한울로보틱스사의 HRB-MTC-B10 모션보드와 디앤지사의 IG-32PGM 01TYPE DC모터[13]로 구성되고 아바타로봇은 OPRoS 시뮬레이터의 velocitymotor 가상장치를 사용하여 구성되며, 실제로봇과 아바타로봇 모두 차륜방식으로 구동된다.

또한 아바타로봇의 경우에는 실제로봇과 달리 LaserRange Finder 가상센서를 부착하여 시뮬레이션 환경에서 거리측정을 가능하게 하였다. 실제로봇과 아바타로봇은 32비트 MS

표 1. 각 로봇모델에 사용된 모듈과 설정변수 값.
Table 1. Module and parameter value used in each robot model.

	실제로봇	아바타로봇
바퀴 제어장치	HRB-MTC-B10 모션보드 12V DC 모터	Velocitymotor
센서	X	LaserRangeFinder
축 지름(m)	0.21	0.21
축 길이(m)	0.46	0.46
반지름(m)	0.3	0.3

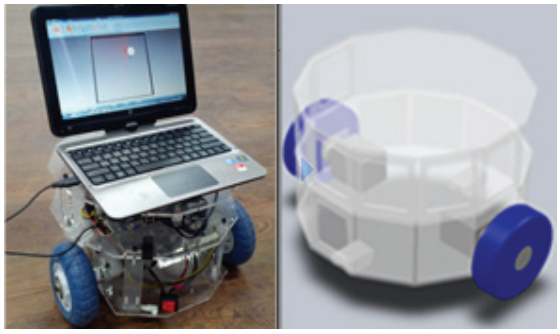

그림 8. 제작된 실제로봇과 아바타로봇.
Fig. 8. Real robot (left) and avatar robot (right).

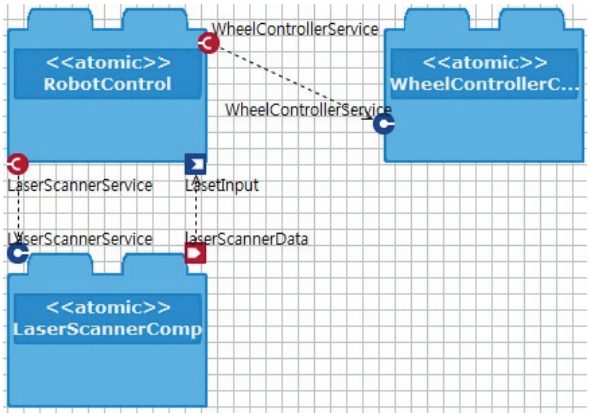

그림 9. 로봇응용 구성.
Fig. 9. Composition of robot application.

윈도우7 운영체제, 4Gb 메모리 환경의 노트북에서 동작한다. 실제로봇과 아바타로봇의 외형은 그림 8과 같으며, 아바타로봇은 실제로봇과 동일한 외형으로 설계되었다.

OPRoS 로봇응용은 이동로봇의 주행과 관련되며, 바퀴제어를 위한 WheelController 컴포넌트와 거리측정을 위한 LaserScanner 컴포넌트, 그리고 WheelController 컴포넌트와 LaserScanner 컴포넌트를 이용하여 장애물을 회피하기 위한 RobotControl 컴포넌트로 구성되며 그림 9에 관련 연결이 표시되어 있다.

로봇응용의 컴포넌트 중 LaserScanner 컴포넌트는 아바타로봇에 부착된 가상 LRF 센서를 통해 가상환경에서 측정한 거리정보를 입력 받으며 이 정보를 이용하여 실제로봇의 주행을 제어한다. 즉 실제로봇의 주행과 동기화된 아바타로봇은 가상환경에서 동작하므로, 아바타로봇의 LRF 센서에서 측정한 거리정보를 이용하여 실제로봇의 주행을 제어하면 실제로봇은 시뮬레이션의 환경에서 주행하는 것과 동일하다고 할 수 있다. 또한 시뮬레이션 도구로는 OPRoS 시뮬레이터[14]를 사용하였으며, 시뮬레이션에서 아바타 로봇을 동작시킨다. 예제로 사용될 HIL 드라이버는 실제로봇과 아바타로봇의 주행을 동기화 하기 위한 HIL_WheelController이며 OPRoS 공통 H/W 추상화 인터페이스 중 WheelController를 상속하여 구현된다. 따라서 OPRoS WheelController 컴포넌트는 일반적인 디바이스 드라이버를 적재하는 방식과 동일하게 HIL 드라이버를 적재할 수 있으며, HIL 드라이버의 실제 H/W 제어모듈을 이용하여 실제로봇의 주행을 제어할 수 있다.

HIL_WheelController 초기화 시에 설정되는 속성값은 WheelController 컴포넌트 프로파일에 입력되는 속성정보이며 그림 10과 같다.

HIL_WheelController에서는 실제장치 제어모듈과 가상장치 제어모듈이 동시에 동작하기 때문에 실제로봇 H/W 구동을 위한 그림 2의 컴포넌트 프로파일과 비교하여 시뮬레이션 접속을 위한 정보와 아바타로봇의 장치정보 등이 추가된다. 프로파일에서 물리장치관련 정보는 실제 H/W 제어모듈에 설정되며, 아바타로봇의 장치정보와 포트 및 ip 등의 정보는 가상 H/W 제어모듈에 설정된다.

WheelController 컴포넌트는 초기화 시점에 드라이버 S/W인 HIL_WheelController를 적재하며, 컴포넌트 프로파일 속성정보를 이용하여 HIL_WheelController를 초기화한다. HIL_Wheel Controller에서는 실제로봇의 하드웨어와 아바타로봇의 하드웨어를 동시에 접근 및 제어하므로, 그림 11과 같이 HIL_ WheelController의 초기화 시점에 실제로봇과 아바타로봇의 H/W 제어모듈을 생성하며 WheelController 컴포넌트 프로파일의 속성정보를 이용하여 초기화한다.

로봇 H/W 제어모듈의 초기화 시에 설정되는 속성값은 그림 10과 같이 OPRoS 컴포넌트의 프로파일에 입력되는 속성정보이며 실제로봇 H/W 제어모듈에 설정될 속성정보는 물리장치관련 정보이다. 이와 달리 아바타 로봇 H/W 제어모듈

```
<properties>
<!—Start of H/W Driver property-->
<!--컴포넌트에서 사용한 디바이스 드라이버 파일의 정보-->
<property name="APIName" type="string">HIL_Driver.dll</property>
<!--장치 축 지름-->
<property name="WheelDiameter" type="double">0.13</property>
<!--장치 축 길이-->
<property name="AxleDistance" type="double">0.30</property>
<!--장치 분산 길이-->
<property name="VarianceDistance" type="double">0.05</property>
<!--장치 분산 방향-->
<property name="VarianceDirection" type="double">1.0</property>
<!--장치 반지름-->
<property name="SafeRadius" type="double">0.3</property>
<!--장치 최고속도-->
<property name="MaximumVelocity" type="double">0.2</property>
<!--장치 가속도-->
<property name="Acceleration" type="double">0.5</property>
<!—Start of Simulation Driver property-->
<!- 아바타로봇 우측 모터 이름-->
<property name="R_Motor_Name" type="string">RightWheel</property>
<!-- 아바타로봇 좌측 모터 이름-->
<property name="L_Motor_Name" type="string">LeftWheel</property>
<!-- 아바타로봇 우측 엔코더 이름-->
<property name="R_Enc_Name" type="string">enc_right</property>
<!-- 아바타로봇 좌측 엔코더 이름-->
<property name="L_Enc_Name" type="string">enc_left</property>
<!--시뮬레이션 접속 IP 주소-->
<property name="ip" type="string">127.0.0.1</property
<!--시뮬레이션 접속 포트-->
<property name="port" type="int">5500</property
<!—아바타 로봇이름-->
<property name="RobotName" type="string">avatar</property>
<!- 동기화 수행 주기-->
<property name="Period" type="string">100</property>
<property name="VelocityStep" type="double">0.01</property>
</properties>
```

그림 10. RILS용 WheelController 컴포넌트 프로파일의 예.
Fig. 10. Example of WheelController Component profile for RILS.

```
int HIL_Driver::InitializeWheelController(Property parameter)
{
    //실제로봇 H/W 제어모듈 생성
    HW_WheelControl = new KNU_HRC01_MC_V20();
    //프로파일정보를 이용한 실제 H/W 제어모듈 초기화
    HW_WheelControl->Initialize(parameter);
    //아바타로봇 H/W 제어모듈 생성
    Sim_WheelControl = new Simulator_WheelComp()
    //프로파일정보를 이용한 가상 H/W 제어모듈 초기화
    Sim_WheelControl->Initialize(parameter);
}
```

그림 11. 드라이버 HIL_WheelController 초기화 예.
Fig. 11. Example of Driver HIL_WheelController initialization.

```
VelocityStep←0.01 //Velocity Step 정보 입력
avatarLinearVelocity←RealLinearVelocity  //실제로봇 속도정보 입력
avatarAngularVelocity←RealAngularVelocity  //실제로봇 각도정보 입력
realBotPos ← HwWheelGetPosition  //실제로봇 위치정보 입력
avatarBotPos ← SimWheelGetPosition  //아바타로봇 위치정보 입력
If is First then
  //실제로봇과 아바타로봇에 동일한 속도 및 각속도로 주행
  SimWheelDriveWheel(avatarLinearVelocity, avatarAngularVelocity)
end
otherwise
  //실제로봇과 아바타로봇의 위치차이 계산
  curPosDIff ← calcPosDiff(RealBotPos, avatarBotPos)
if avatarBot is faster than RealBot
    //실제로봇과 아바타로봇의 각속도가 다른경우
    if curPosDiff.theta is not equal 0
      //아바타로봇 정지 및 각속도 재설정
      SimWheelDriveWheel(0,0)
      SimWheelDriveWheel(0,curPosDiff.theta)
    Otherwise
      //아바타로봇 정지
      SimWheelDriveWheel(0, 0)
  prePosDiff ← curPosDiff
 end
Otharwise
    //거리차이가 이전보다 더 커진경우
    If curPosDiff greater than prePosDiff
      //아바타로봇의 속도증가
      avatarLinearVelocity ← avatarLinearVelocity + VelocityStep
    Otherwise
      //아바타로봇의 속도감소
      avatarLinearVelocity ← avatarLinearVelocity - VelocityStep
   avatarAngularVelocity ← curPosDiff.theta
   //아바타로봇 주행
   SimWheelDriveWheel(avatarLinearVelocity, avatarAvgularVelocity)
   prePosDiff ← curPosDiff
end
```

그림 12. 실제로봇과 아바타로봇의 주행동기화 알고리즘.
Fig. 12. Synchronization algorithm of robot and avatar robot.

에 설정되는 속성정보는 아바타 로봇의 가상장치관련 정보와 시뮬레이션과의 통신을 위한 정보이다.

OPRoS 로봇응용의 수행에 따라 실제로봇과 아바타 로봇이 동작한다. 하지만 실제로봇과 아바타 로봇은 환경에 따라 다르게 동작할 수 있음으로, 실제로봇의 정보를 기반으로 아바타 로봇의 동작을 보정하기 위한 알고리즘이 개발되었으며 그림 12와 같다.

먼저, 실제로봇과 아바타로봇이 처음 동작하는 시점에서는 실제로봇에 설정된 속도정보와 동일한 속도로 아바타로봇을 동작시킨다. 일단 실제로봇과 아바타 로봇이 동작한 이후에는 그림 5와 같이 동작의 차이가 발생할 수 있으므로 보정작업을 수행해야 한다. 여기서 보정 알고리즘은 실제로봇과 아바타로봇 간의 위치정보 오차를 기반으로 아바타로봇의 동작을 결정한다. 만약 오차계산 결과로 아바타로봇이 실제로봇보다 선행하는 경우 선속도와 각속도를 조정하여 아바타로봇의 일시적으로 정지시키거나 각속도를 변경한다. 이와 반대로 아바타로봇이 실제로봇보다 후행하는 경우에는 선속도에 일정 단위속도인 VelocityStep 값을 가감하여 선속도를 결정한다. 선행하는 실제로봇과의 차이가 이전 보정시보다 더 커진 경우 선속도에 VelocityStep 값을 증가시켜 아바타로봇의 속도를 증가시키고, 반대의 경우에는 감소시킨다. 아바타로봇이 후행하는 경우에는 VelocityStep을 가감하여 결정된

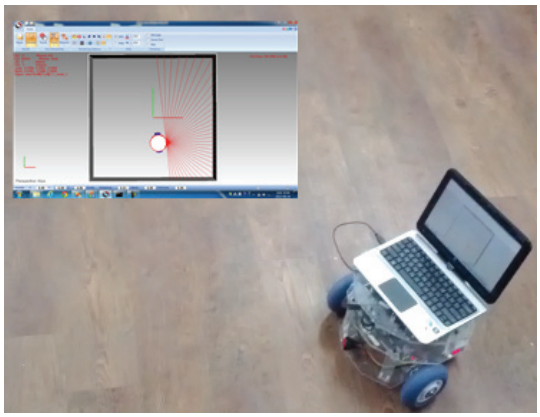

그림 13. RILS의 수행.
Fig. 13. Execution of RILS.

선속도와 계산된 각속도차이를 이용하여 동작을 변경함으로써 실제로봇의 동작에 근접할 수 있다. 여기서 Velocity Step은 사용자가 임의로 초기에 설정한 값이며 아바타로봇과 실제로봇간의 이동차이를 빠르게 보상할 수 있도록 설정되어야 한다. 그림 12의 주행동기화 알고리즘은 그림 10의 컴포넌트 프로파일 속성정보에 입력된 주기정보에 따라 100ms 마다 주기적으로 동작하며, VelocityStep은 0.01(m/s)로 설정하였다. 오차보정 알고리즘은 실제로봇과 아바타로봇의 위치정보를 이용하여 실제로봇의 동작을 기준으로 아바타로봇을 보정한다.

실제로봇과 아바타로봇이 RILS를 수행하며 동작하는 상황이 그림 13에 표시되어 있다. OPRoS 로봇응용의 수행에 따라 실제로봇과 아바타로봇은 각각 실제환경과 가상환경에서 동시에 주행을 시작하며, 동작중인 실제로봇은 가상환경 상의 장애물 측정정보를 이용해 장애물을 회피하여 동작한다. 또한 아바타로봇은 주행동기화 알고리즘을 통해 시뮬레이션에서 실제로봇의 동작을 재현한다. RILS가 동작하는 영상은 youtube 영상[15] 에서 확인할 수 있다.

동기화를 사용한 제안된 방법을 기반한 아바타 로봇과 실제로봇을 동작시킨 결과는 거리 차이들이 그림 14, 표2, 그림 15 및 표 3에 표시되어 있다. 그림 14, 그림 15, 표 2 그리고 표 3에서 표시된 원 번호 (예 : ①)는 사용되는 속도 프로파일을 표시한다.

그림 14, 그림 15. 표2 및 표3을 보면 속도가 빠를 때 오차가 커짐을 알 수 있다. 속도가 0.2 m/s에서 속도가 0.6 m/s로 변할 경우 이동 거리 오차는 약 2배 정도 커지지만 1.1cm에서 2.5cm로 커지기 때문에 표 1의 로봇 크기를 비교하면 오차는 어느 정도 무시할 만하다. 또한 일정한 속도로 주행하는 속도 프로파일(그림 14)과 주행속도를 변경하는 속도 프로파일(그림 15)의 거리 오차를 살펴보면 차이가 거의 없다는 것을 알 수 있다. 그림 5에서 실제로봇과 가상로봇이 주행명령을 수행 할 시에 실제로봇과 가상로봇 사이에는 분명한 동작의 차이가 발생하지만, 제안된 방법을 구현한 RILS를 사용한 경우에는 아바타로봇의 동작이 실제로봇 정보에 기반하여 수행되므로 그림 14에서 볼 수 있듯이 실제로봇과 유사하게 동작한다. 위치보정 알고리즘이 적용되기 전인 그림 5의 시뮬레이션을 수행 시 실제로봇과 가상로봇의 평균 거리 차이는 0.070(m) 분산 0.00028이며, RILS 적용 후 표 2에서

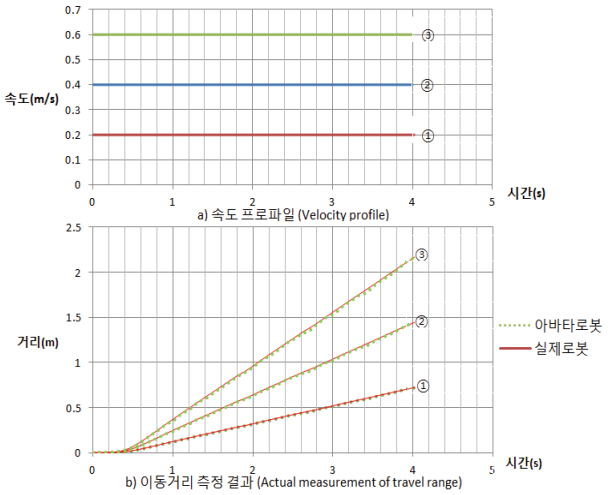

그림 14. 일정 속도 하에서 실제 로봇과 아바타 로봇의 이동 거리 비교.

Fig. 14. Comparison of Moving Distance between the robot and the avatar robot with the constant moving speed.

표 2. 그림 14의 실제로봇과 아바타 로봇의 거리 차이.

Table 2. Distance difference between robot and avatar robot in Fig. 14.

속도 프로파일 번호	거리 차이	
	평균 (단위: m).	분산
①	0.011	0.00075
②	0.016	0.00094
③	0.025	0.00113

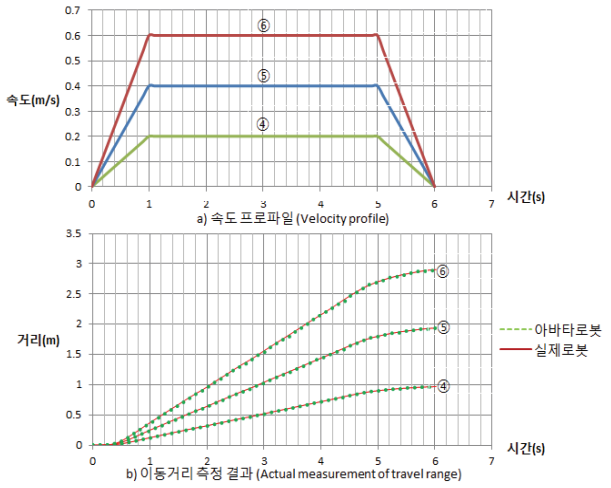

그림 15. 속도 프로파일 하에서 실제 로봇과 아바타 로봇의 이동거리 비교.

Fig. 15. Comparison of Moving Distance between the robot and the avatar robot with speed profile.

표 3. 그림 15의 실제로봇과 아바타 로봇의 거리 차이.

Table 3. Distance difference between robot and avatar robot in Fig. 15.

속도 프로파일 번호	거리 차이	
	평균 (단위: m).	분산
④	0.010	0.00071
⑤	0.015	0.00091
⑥	0.023	0.00109

확인할 수 있듯이 평균 거리 차이는 약 0.011(m) 분산 0.00075로써 RILS 수행 시 실제로봇과 더 근접하게 동작한다는 것을 확인할 수 있다. 그림 15은 속도 정보를 시간에 따라 가속도를 주는 형태로 적용하였지만 그림 14와 유사한 성능을 볼 수 있었다.

하지만 본 동기화 알고리즘은 고정된 VelocityStep을 사용하므로 주행중인 실제로봇의 속도 변화가 큰 경우에 한계점을 갖는다. 예를 들어 VelocityStep이 작다면 속도 변화가 적은 경우 보정이 유효하게 동작할 수 있지만 설정된 VelocityStep 값으로 수용할 수 없는 속도 변화가 발생한다면 오차가 누적되어 보정이 원활히 수행될 수 없으므로 속도 및 오차에 따라 VelocityStep 값을 변화시킬 수 있는 방법이 요구된다.

V. 결론

본 논문에서는 RILS에서 실제로봇과 아바타로봇 이동 거리의 차이를 보정하기 위한 RILS 구조를 제안하였으며 OPRoS 로봇시스템에 구현하여 유효성을 검증하였다. 제안하는 방법은 HIL 드라이버를 생성하여 실제로봇과 아바타로봇에서 발생할 수 있는 차이를 지속적으로 감지하며, 실제로봇의 H/W 정보를 기반으로 아바타로봇의 동작을 보정한다. 기존연구[12]에서는 환경과 기계적 특성에 따라 실제로봇과 아바타로봇의 동작 차이가 발생하며 중첩될 시에 실제/가상 로봇이 서로 다르게 동작할 수 있었지만, 본 논문에서 제시한 RILS는 아바타로봇이 실제로봇의 정보에 기준하여 주기적으로 보정됨으로써 아바타로봇이 실제로봇과 근접하게 동작한다. 실험을 통해 RILS 적용 후 실제로봇과 아바타로봇의 오차가 감소하였으며 RILS의 유효성을 확인할 수 있었다.

향후 동기화 기능을 일반화시킬 수 있는 메커니즘이 개발되어야 할 것이며, VelocityStep을 선정할 수 있는 방법에 대한 연구가 필요하다.

참고문헌

[1] M. S. Kim, S. H. Han, and H. S. Park, “An automatic robot software design tool: An open platform for robotics services,” *Emerging Technologies and Factory Automation (ETFA)*, Bilbao, Spain, pp. 1-4, Sep. 2010.

[2] D. Fennibay, A. Yurdakul, and A. Sen, “Introducing hardware-in-the-loop concept to the hardware/software co-design of real-time embedded system,” *Computer and Information Technology (CIT)*, Bradford, UK, pp. 1902-1909, Jun. 2010.

[3] M. Hillenbrand, M. Heinz, and K. D. Muller-Glaser, “Rapid specification of hardware-in-the-loop test systems in the automotive domain based on the electric / electronic architecture description of vehicles,” *Rapid System Prototyping (RSP)*, Fairfax, CA, USA, pp. 1-6, Jun. 2010.

[4] Q. Wu, L. Lei, J. Chen, and W. Wang, “Research on hardware-in-the-loop simulation for advanced front-lighting system,” Computational Intelligence and Industrial Application, Wuhan, China, pp. 671-675, Dec. 2008.

[5] D. Michalek, C. Gehsat, R. Trapp, and T. Bertram, “Hardware-in-the-loop-simulation of a vehicle climate controller with a combined HVAC and passenger compartment model,” *Advanced Intelligent Mechatronics, Monterey*, CA, USA, pp.

1065-1070, Jul. 2005.
[6] W. J. Hwang and C. G. Kang, "HILS of the braking system of high speed train," *Journal of Institute of Control, Robotics and Systems (in Korean)*, vol. 7, no. 1, pp. 432-438, May 2001.
[7] S. K. Hong and S. H. Back, "A study on flight trajectory generations and guidance/control laws: validation through HILS," *Journal of Institute of Control, Robotics and Systems (in Korean)*, vol. 14, no. 12, pp. 1238-1243, Dec. 2008.
[8] M. Linjama, T. Virvalo, J. Gustafsson, J. Lintula, V. Aaltonen, and M. Kivikoski, "Hardware-in-the-loop environment for servo system controller design, tuning, and testing," *Microprocessors and Microsystems*, vol. 24, no. 1, pp. 13-21, Mar. 2000.
[9] S. Oncu, S. Karaman, L. Guvenc, S. S. Ersolmaz, E. S. Ozturk, E. Cetin, and M. Sinai, "Robust yaw stability controller design for a light commercial vehicle using a hardware in the loop steering test rig," Intelligent Vehicles Symposium, Istanbul, Turkey, pp. 852-859, Jun. 2007.
[10] I. G. Jang, I. k. Seo, J. W. Jeon, and S. H. Hwang, "Development of hardware-in-the-loop simulator for testing embedded system of automatic transmission," *Journal of Institute of Control, Robotics and Systems (in Korean)*, vol. 14, no. 3, pp. 301-306, Mar. 2008.
[11] A. Martin and M. R. Emami, "Analysis of robotic hardware-in-the-loop simulation architecture," *Intelligent Robot and Systems*, San Diego, CA, USA, pp. 46-51, Oct. 2007.
[12] X. Hu, "Applying robot-in-the-loop-simulation to mobile robot systems," *International Conference on Advanced Robotics*, Atlanta, GA, USA, pp. 506-513, Jul. 2005.
[13] http://www.dnj.co.kr/_data/2_dcgm/pnig_spec_ig32p01.htm.
[14] http://ropros.org/display/documentation/OPRoS+Manual.
[15] http://www.youtube.com/watch?v=mfnVWzfIcmo&feature=plcp.
[16] S. Kim, S. Cho, S. Cho, and C. Park, "Development of operation network system and processor in the loop simulation for swarm flight of small UAVs," *Journal of Institute of Control, Robotics and Systems (in Korean)*, vol. 18, no. 5, pp. 433-438, May 2012.
[17] C. Hong and H. S. Park, "Design and implementation of web-based software engineering tool for robot," *Journal of Institute of Control, Robotics and Systems (in Korean)*, vol. 17, no. 9, pp. 908-915, Sep. 2011.

김 성 훈

2009년 강원대학교 전자통신공학과 졸업. 2010년~현재 동 대학원 석사 과정. 관심분야는 로봇 S/W 플랫폼, 로봇 시뮬레이션.

박 홍 성

1983년 서울대학교 제어계측공학과 학사 졸업. 1986년 동 대학원 석사. 1992년 동 대학원 박사. 1992년~현재 강원대학교 전자통신공학과 교수. 관심분야는 로봇 S/W 플랫폼, 무선데이터통신, 실시간 통신.