Journal of Korea Robotics Society (2016) 11(4):270-276
https://doi.org/10.7746/jkros.2016.11.4.270 ISSN: 1975-6291 / eISSN: 2287-3961

모듈러 로봇의 작업 적응성을 위한 자가 재구성 제어 엔진

Design of Self-Reconfigurable Kinematics and Control Engine for Modular Robot

도 현 민[†], 최 태 용[1], 박 동 일[1], 김 두 형[1], 손 영 수[1]

HyunMin Do[†], Tae-Yong Choi[1], DongIl Park[1], DooHyeong Kim[1], Youngsu Son[1]

Abstract This paper proposes a design methodology of self-reconfigurable kinematics and control engine for modular and reconfigurable robots. A modular manipulator has been proposed to meet the requirement of task adaptation in versatile needs for service and industrial robot area and the function of self-reconfiguration is required to extend the application of modular robots. Kinematic and dynamic contexts are extracted from the module and assembly information and related codes are automatically generated including controller. Thus a user can easily build and use a modular robot without professional knowledge. Simulation results are presented to verify the validity of the proposed method.

Keywords Self-reconfiguration, Modular robot, Task adaptation

1. 서 론

최근 IT제품 등 소형 전자제품을 중심으로 한 생산 현장에서 다품종 소량 생산 중심의 작업이 많이 이루어지고 있고, 수작업에 의존하는 기존 공정을 자동화하기 위하여 이러한 생산 방식에 적합한 로봇의 수요가 증가하고 있다. 다품종 소량 생산 방식에서 로봇을 적용한 자동화를 위해서는 규격화된 단순 작업만이 아니라 작업 대상물의 변경과 현장의 수요에 맞추어 빠르고 효과적으로 작업할 수 있는 로봇의 작업 적응성이 필요하다. 이러한 작업 적응성 요구를 만족시키기 위한 해결방안으로 자가 재구성이 가능한 모듈러 로봇이 좋은 해결방안이 될 수 있다. 이러한 작업 적응성 요구는 생산현장의 제조용 로봇뿐만 아니라 다양한 수요가 존재하는 서비스 로봇에서도 그 필요성이 증가하고 있고 향후 많은 수요가 예상된다.

로봇의 작업 적응성을 크게 두 가지로 구분하면, 하나는 주어진 작업에 적합하도록 로봇의 구조를 구성할 수 있는 하드웨어적인 측면이 있고, 다른 하나는 환경에 맞는 로봇의 구조가 결정되어 조립이 된 후 사용자가 전문전인 지식을 가지고 있지 않더라도 쉽게 로봇을 사용할 수 있도록 하는 소프트웨어 측면이 있다. 하드웨어측면에서는 다양한 로봇의 구조를 구성할 수 있도록 구동모듈과 링크를 모듈화, 시리즈화하여 다양한 라인업을 구성하는 것이 필요하다. 이러한 모듈의 라인업에서 필요한 모듈을 골라서 사용자가 쉽게 로봇을 재구성할 수 있다. 소프트웨어 측면에서는 사용자가 재구성한 로봇 팔을 전문적인 지식이 없는 사용자가 쉽게 사용할 수 있도록 하기 위한 자가재구성 기능이 필요하다. 즉, 로봇이 구성되면 하드웨어로부터 필요한

Received : Sep. 13. 2016; Revised : Oct. 31. 2016; Accepted : Oct. 31. 2016
※This work was supported by the Ministry of Trade, Industry & Energy and the Korea Evaluation Institute of Industrial Technology (KEIT) under program number 10048920.
[†]Corresponding author: Department of Robotics and Mechatronics, Korea Institute of Machinery and Materials, Daejeon, 34103, Korea (hmdo@kimm.re.kr)
[1]Department of Robotics and Mechatronics, Korea Institute of Machinery and Materials, Daejeon, 34103, Korea

파라미터를 추출하고 이로부터 로봇 구동을 위한 기구학/동역학 정보를 생성하고 제어엔진을 자동적으로 구성함으로써 쉽고 빠르게 로봇을 사용할 수 있다. 모듈러 로봇의 경우 사용자가 구성할 수 있는 모듈의 조합이 너무 많기 때문에 모듈 공급자(생산자)가 모든 모듈 조합에 대하여 제어 엔진을 사전에 구현해 놓을 수가 없으므로 소프트웨어 측면에서의 자가재구성이 반드시 필요한 기능이라고 할 수 있다. 본 논문에서는 이러한 소프트웨어 측면에서의 자가 재구성 기능을 중점적으로 논하고자 한다.

자가 재구성이 가능한 모듈러 로봇 시스템에 대해서 다양한 연구가 이루어져 왔는데[1,2], 주로 하드웨어의 자가재구성 방법과 모듈러 로봇의 특성을 고려한 제어 기법에 관한 내용이다[3,4,5,6]. 모듈러 로봇은 자가 재구성 단계에 따라서 크게 자동 조립(self-assembly), 자동 재구성(self-configuring), 수동 재구성(manual-configuring)의 세 가지 타입으로 분류할 수 있다[7]. 자동 조립(self-assembly)은 로봇 스스로 구조를 바꿀 수 있는 형태로 가장 높은 레벨의 자가재구성 단계로 볼 수 있다. 자동 재구성(self-configuring)은 하드웨어의 경우 사람의 도움을 받아 재구성하여 조립해야하지만, 소프트웨어는 사용자가 작업정보만을 입력하면 로봇 구동에 필요한 정보를 로봇 스스로 재구성하는 단계이다. 수동 재구성(manual-configuring)은 하드웨어, 소프트웨어를 포함하여 모든 재구성이 사람의 도움으로 이루어지는 단계이다.

본 논문은 사용자가 제공된 하드웨어 모듈을 이용하여 사용 목적에 맞도록 구조를 구성한 모듈러 로봇을 쉽고 빠르게 사용하기 위하여 기존 연구와 달리 기구학을 포함하는 제어 엔진을 자동으로 구성하는 방법을 제안하고 있다. 제안한 방법을 이용하면 사용자는 간단한 목표 작업의 설정을 통해 모듈러 로봇 구성부터 제어까지 간편하게 구현이 가능해 짐으로서 비전문가들도 다양한 작업에 로봇을 적용시킬 수 있는 장점이 있다. 또한 위의 자가 재구성 분류단계에 따르면 자동 조립과 자동 재구성 단계의 모듈러 로봇에서 반드시 필요한 내용이라고 할 수 있다.

2장에서는 제어 엔진의 자가 재구성을 위하여 하드웨어로부터 필요한 정보를 추출하고 이로부터 기구학 및 제어 엔진 정보를 자동적으로 구성하는 방법을 제안하고, 3장에서는 자동 재구성된 제어 엔진의 검증을 위한 시뮬레이션을 수행하고 그 결과를 제시하였다. 끝으로 4장에서는 결론 및 추후 연구 방향에 대하여 기술한다.

2. 제어 엔진 자가 재구성

2.1 자가 재구성을 위한 모듈라 로봇 구성 정보 추출

사용자가 필요한 작업을 수행하기 위하여 주어진 모듈을 조합하여 원하는 형태의 모듈라 로봇을 구성하게 되므로 사전에 모듈라 로봇의 구성에 관한 정보를 획득할 수가 없다. 따라서 구성된 모듈라 로봇이 주어질 경우 이에 적합한 제어 엔진을 자동적으로 구성하기 위하여 우선 모듈라 로봇의 연결 정보를 파악해야 한다. 모듈은 생산 단계에서 각 모듈별 기구학 특성과 동역학 특성 정보의 파악이 가능하므로 연결된 모듈의 정보를 알면 해당 정보의 파악이 가능하고 또한 연결정보를 확인하기 위하여 체결 조인트의 위치 및 체결 방향의 정보가 필요하다.

각 모듈로부터 필요한 정보를 받아오기 위하여 각 모듈에서 제공해야 할 모듈 데이터의 구조를 정의하였다. 이 정보는 모듈의 생산 단계에서 제공 하는 정보로 모듈 ID, 모듈 내 조인트들의 위치를 표시한 기구학 정보, 그리고 질량과 무게 중심, 관성치의 동역학 파리미터로 구성된다. 이 때 각 모듈의 기준원점의 가상의 바디 좌표계를 기준으로 하여 정보를 표시한다. 구동 모듈과 이에 해당하는 모듈 데이터의 예를 Fig. 1과 Table 1에 제시하고 있다.

연결된 각 모듈의 정보를 모듈 데이터 파일을 통하여 획득한 후, 모듈라 로봇의 모듈간 연결 토폴로지를 기술하기 위하여 어셈블리 데이터의 구조를 정의하였다. 여기에는 연결순서에 따라 부모 모듈과 자식 모듈을 구분하고 연결된 조인트 정보, 조인트 간 상대적인 위치를 나타내는 좌표계 정보 및 회전 축 정보 등을 표시한다. 그리고 모듈라로봇의 특성상 모듈간 체결시 구동을 위한 연결인지 아니면 단순히

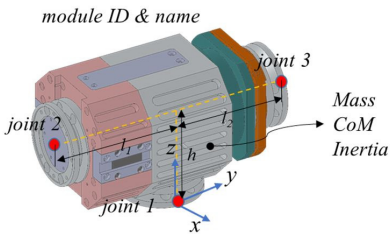

Fig. 1. Actuator module (example)

Table 1. Structure of module data

item		Value (example)
module ID		001
module name		Actuator1
joint 1	position	x='0', y='0', z='0'
	property	assembly
	axis	'z'
joint 2	position	x='0', y='-l1', z='h'
	property	joint
	axis	'xyz'
mass		10.99
com		x='0', y='0', z='h'
inertia		lxx='0.4' lyy='0.4' lzz='0.1' lxy='0' lyz='0' lxz='0'

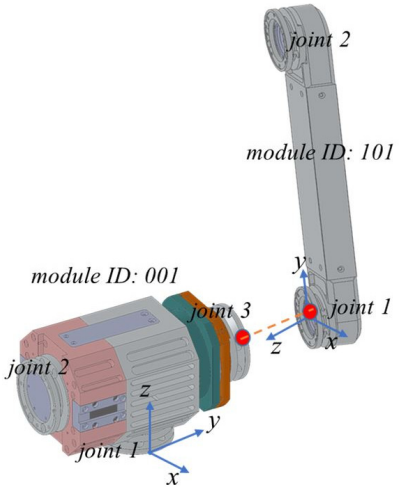

Fig. 2. Assembly of actuator module and link module (example)

Table 2. Structure of assembly data

item	value (example)
parent ID	001
child ID	101
parentAssemblyJoint	joint3
childAssemblyJoint	joint1
parentAxis1	'x'
childAxis1	'x'
parentAxis2	'y'
childAxis2	'-z'
rotationAxisZ	'z'
rotationAxisX	'x'
isFixed	'false' (or 'true')

고정을 위한 연결인지를 표시하는 정보(isFixed)를 포함하도록 하였다. 연결부를 fixed로 설정할 경우 기구학적으로 고정하는 효과를 가지는데, 모듈라 로봇이기 때문에 로봇의 조립과정에서 필요에 따라 고정해야 하는 경우가 발생할 수 있다. 이러한 어셈블리 데이터의 예를 Fig. 2와 Table 2에 제시하였다.

이러한 모듈 데이터를 자동적으로 획득하게 위하여 해당 정보를 받을 수 있는 프로토콜을 모듈과 상위 제어기간에 정의를 한 후 상위 제어기에서 요청을 하게 되면 Fig. 3과 같이 각 모듈에서 해당 정보를 상위의 로봇 제어기로 보내어 해당 정보를 추출하고, 이를 근거로 어셈블리 데이터를 생성한다.

2.2 기구학 및 동역학 자가 재구성

모듈 데이터 정보와 어셈블리 데이터 정보를 바탕으로 Fig. 4와 같이 기구학 및 동역학을 자동적으로 구성한다. 일반적인 N개의 바디로 이루어진 기구학 및 동역학 식은 추상화되기 때문에 static하게 고정되고, 모듈 구성에 따라 변화하는 부분은 kinematic context와 dynamic context이다. 우선 모듈 데이터 정보로부터 연결 조인트의 상대적인 거리 정보(displacement)를 추출할 수 있고 또한 어셈블리

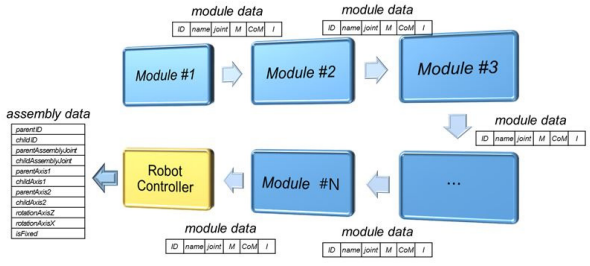

Fig. 3. Flow of module data

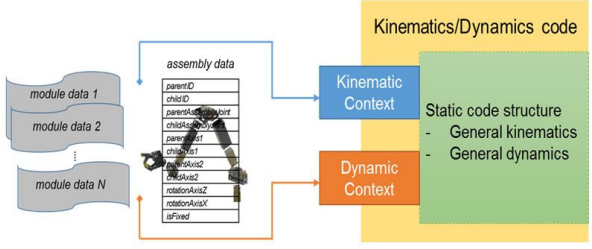

Fig. 4. Extraction of kinematic and dynamic contexts

데이터로부터 연결 조인트간의 자세 정보(rotation) 및 연결 순서 정보를 추출할 수 있어 이로부터 DH 정보를 계산한다. 이렇게 계산된 정보로부터 기구학 연산 코드를 자동적으로 생성한다.

기구학 연산은 6자유도 이하의 deficient kinematics와 6 자유도 kinematics, 그리고 6자유도 이상의 redundant kinematics로 구분하여 수행한다. 즉, 로봇의 자유도에 따라 이에 맞는 기구학 연산 코드를 생성한다. 그리고 각 모듈에 대한 동역학 파라미터를 모듈 데이터 정보에서 얻을 수 있으므로 기 계산된 기구학 정보를 근거로 로봇의 동역학 연산 코드를 자동적으로 생성한다. Fig. 5는 생성된 기구학 및 동역학 연산 코드의 예를 제시하고 있다. 기구학 연산을 위하여 필요한 오프셋 정보를 추출하여 반영하고 모듈간 연결정보가 표시되어 있음을 확인할 수 있다. 또한 동역학 연산을 위한 질량, 무게중심, 관성치 정보가 반영되어 있다.

2.3 제어 엔진 자가 재구성

기구학 및 동역학 자가 재구성이 완료되면 이로부터 로봇 제어에 필요한 제어 엔진의 자가 재구성이 가능하다. 우선 로봇의 자유도에 따라서 각 조인트별 제어기를 구성한다. 이 때 제어 게인은 기본값으로 주어지지만 성능향상을 위해서는 하드웨어 특성에 맞는 제어 게인 설정이 필요하다. 로봇 제어에 관한 전문적인 지식이 없는 경우 제어 게인 설정이 어려울 수 있어 이 경우에는 자동 게인 튜닝 알고리즘[8]을 적용하여 게인 튜닝을 진행할 수 있다. 이는 각 모듈의 물리적 파라미터를 근거로 초기 게인을 설정하고, 로봇의 전체 관절을 정해진 궤적에 따라 움직이면서 PID 제어기의 위치, 속도, 적분 게인을 찾아나가는 절차에 의하여 진행된다. 자동 게인 튜닝에 관한 상세한 내용은 본 논문의 범위를 벗어나는 내용으로 상세 내용을 기술하지는 않겠다.

그리고 자동 생성된 기구학 연산 코드를 적용하여 작업 공간상에서 로봇 말단의 위치 제어를 수행할 수 있어, 사용자가 조인트 공간상에서 뿐만 아니라 작업 공간상에서의 로봇 구동도 간단하게 수행할 수 있다. 또한 동역학 연산 코드를 적용하여 동역학 기반의 제어기 구성도 가능하다. 따라서 사용자가 로봇 기구학 및 동역학 구성을 위하여 별도의 작업을 할 필요 없이 필요한 제어 로직을 추가하여 제어기를 쉽게 구성할 수 있고, 로봇 제어에 관한 전문 지식이 없는 사용자도 자동적으로 구성되는 제어 엔진을 사용하여 간단히 모듈라 로봇의 모션 제어를 수행할 수 있다. 기구학/동역학을 포함한 전체적인 제어 엔진의 자가 재구성 작업 흐름은 Fig. 6과 같다.

3. 시뮬레이션을 통한 검증

앞 절에서 제안한 자가 재구성 방법을 검증하기 위하여 주어진 모듈을 적용하여 모듈라 로봇을 구성하고 모션 제어 시뮬레이션을 수행할 수 있는 시뮬레이터를 개발하였다. 시뮬레이터에서는 사용자가 개별 모듈을 모델링하여 시뮬레이터에 입력하고, 이 모듈을 사용하여 로봇을 구성할 수 있는 인터페이스가 제공된다. 본 논문에서는 2개의 구동 모듈과 2개의 링크 모듈을 모델링하여 시뮬레이션을 수행하였다. 1개의 구동 모듈은 Fig. 1과 같이 3개의 조인트를 가지는 1자유도 모듈이고, 다른 1개는 2개의 조인트를 가지는 1자유도 모듈이다. 링크 또한 1개는 Fig. 2와 같이 2개의

```
// for kinematics
...
Offset[0].set(Rotation(1,0,0,0,1,0,0,0,1), Displacement(0.0000,0.0000,0.1160));
Offset[1].set(Rotation(1,0,0,0,0,1,0,-1,0), Displacement(0.1268,0.0920,0.0920));
Offset[2].set(Rotation(1,0,0,0,1,0,0,0,1), Displacement(0.0000,-0.3000,-0.1545));

_joint_0.setJointFrame(Offset[0]);
_joint_1.setJointFrame(Offset[1]);
_joint_2.setJointFrame(Offset[2]);

connect(0,1,_joint_0);
connect(1,2,_joint_1);
connect(2,3,_joint_2);

// for dynamics
...
inertiaTemp.set(1.715703, 0.00266, 0.000182, 0.060863, 0.002654, 0.005444, 0.00539,
                                                    0.000006, 0.000012, 0.000004);
modelingFrame.set(Rotation(0,0,1,1,0,0,0,1,0), Displacement(0.0634,-0.0000,0.0920));
inertia[1].addInertia(inertiaTemp, modelingFrame);
inertiaTemp.set(0.907541, 0.026174, 0, 0.15, 0.014652, 0.014429, 0.00047, 0, 0, 0);
modelingFrame.set(Rotation(0,-1,0,1,0,0,0,0,1), Displacement(0.1268,0.0920,0.0920));
inertia[2].addInertia(inertiaTemp, modelingFrame);
...
```

Fig. 5. Kinematics and dynamics context related code (automatically generated)

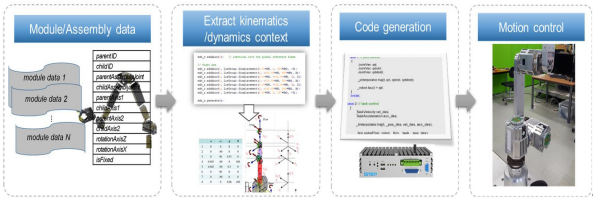

Fig. 6. Overall procedure of the proposed self-reconfigurable kinematics and control engine

actuator module 1

```
<block id="Module1" name="Module1" isBaseBlock="false" isToolBlock="false" tag="Module">
  <pointList>
    <point x="0" y="0" z="0" jointAssembly="assembly" useAxis="xyz"/>
    <point x="-0.0944" y="0" z="0.0634" jointAssembly="assembly" useAxis="x"/>
    <point x="0.092" y="0" z="0.0634" jointAssembly="joint" useAxis="x"/>
  </pointList>
  <mass value="1.715703"/>
  <com x="0.002660" y="0.000182" z="0.060863"/>
  <inertia Ixx="0.002654" Iyy="0.005444" Izz=" 0.005390" Ixy=" 0.000006" Iyz="0.000012" Ixz="0.000004"/>
</block>
```

link module 1

```
<block id="Link1" name="Link1" isBaseBlock="false" isToolBlock="false" tag="Link">
  <pointList>
    <point x="0" y="0" z="0.3" jointAssembly="assembly" useAxis="x"/>
    <point x="0" y="0" z="0" jointAssembly="assembly" useAxis="x"/>
  </pointList>
  <mass value="0.907541"/>
  <com x="0.026174" y="0.000000" z=" 0.150000"/>
  <inertia Ixx="0.014652" Iyy="0.014429" Izz="0.000470" Ixy="0.000000" Iyz="0.000000" Ixz="0.000000"/>
</block>
```

actuator module 2

```
<block id="Module2" name="Module2" isBaseBlock="false" isToolBlock="false" tag="Module">
  <pointList>
    <point x="0" y="0" z="0" jointAssembly="joint" useAxis="x"/>
    <point x="-0.1545" y="0" z="0" jointAssembly="assembly" useAxis="x"/>
  </pointList>
  <mass value="1.148081"/>
  <com x="-0.079752" y="0.000238" z="0.000000"/>
  <inertia Ixx="0.001836" Iyy="0.003043" Izz="0.002778" Ixy="0.000000" Iyz="-0.000001" Ixz="-0.000000"/>
</block>
```

link module2

```
<block id="Link2" name="Link2" isBaseBlock="false" isToolBlock="false" tag="Link">
  <pointList>
    <point x="0" y="0" z="0"/>
  </pointList>
  <mass value="0.174875"/>
  <com x="-0.009221" y="0.000000" z="0.072582" jointAssembly="assembly" useAxis="x"/>
  <inertia Ixx="0.001121" Iyy="0.001066" Izz="0.000091" Ixy="0.000000" Iyz="0.000000" Ixz="0.000002"/>
</block>
```

Fig. 7. Module data file

조인트를 가지고 있어 모듈간 연결용으로 사용이 가능하고, 다른 1개는 1개의 조인트를 가지고 있어 로봇 말단에 사용이 가능하다. 각 모듈에 대한 모듈 데이터 정보는 Fig. 7과 같고, 해당 모듈들을 연결하여 구성한 3자유도 로봇은 Fig. 8과 같다. 우선 구동모듈 1을 2개 연결한 후 링크 모듈 1을 연결하고 여기에 구동모듈 2, 링크 모듈 2의 순서로 연결하였다. 구성된 로봇에서 생성한 어셈블리 데이터는 Fig. 9와 같다. 총 5개의 모듈이 연결되어 있음을 알 수 있고, 이 때 연결축과 연결방향이 표시되어 있다.

시뮬레이션을 수행하기 위하여 3자유도 모듈라 로봇을 시뮬레이터상에서 구현하고 제어엔진 재구성 및 코드 생성 작업을 수행하였다. 모듈 데이터 정보와 어셈블리 데이터 정보로부터 기구학 연산 코드와 동역학 연산 코드를 자동으로 생성하고 이를 모듈라 로봇의 모션 제어에 활용하였다. 즉, 사용자는 로봇 기구학 및 동역학 등 로봇에 관한 전문적 지식이 없이도 필요한 연산 코드를 사용할 수 있고, 로봇의 모션 제어를 위한 기본적인 제어기 코드 또한 자동적으로 생성이 되어 기본 설정 게인을 적용하여 로봇 제어를 수행할 수 있도록 하였다. 동역학 연산 코드 활용을 고려하여

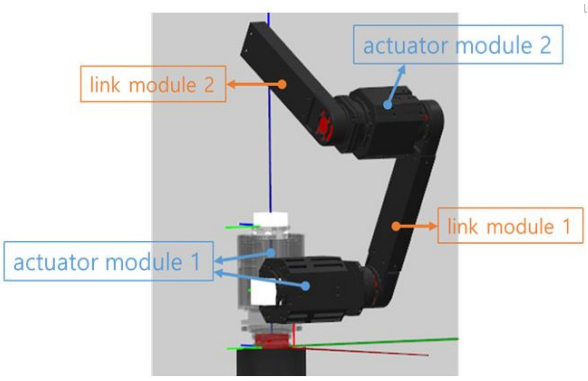

Fig. 8. Assembled 3dof modular robot

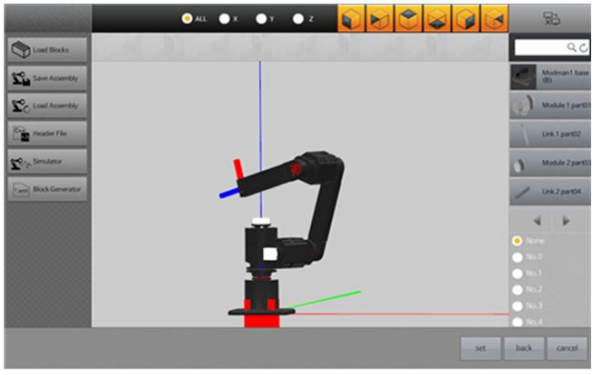

Fig. 10. 3dof modular manipulator model in simulation

```
<part id="1" parentId="0" blockId="Module1" parentAssemblyJointIndex="0" assemblyJointIndex="2" parentAxis1="-z" childAxis1="x"
                          parentAxis2="-x" childAxis2="z" rotationAxisZ="z" rotationAxisX="x" jointOption="0" isFixed="false"/>
<part id="2" parentId="1" blockId="Module1" parentAssemblyJointIndex="0" assemblyJointIndex="0" parentAxis1="-z" childAxis1="z"
                          parentAxis2="-y" childAxis2="x" rotationAxisZ="" rotationAxisX="" jointOption="-1" isFixed="false"/>
<part id="3" parentId="2" blockId="Link1" parentAssemblyJointIndex="2" assemblyJointIndex="1" parentAxis1="x" childAxis1="x"
                          parentAxis2="y" childAxis2="z" rotationAxisZ="y" rotationAxisX="x" jointOption="0" isFixed="false"/>
<part id="4" parentId="3" blockId="Module2" parentAssemblyJointIndex="0" assemblyJointIndex="1" parentAxis1="-x" childAxis1="x"
                          parentAxis2="z" childAxis2="z" rotationAxisZ="" rotationAxisX="" jointOption="-1" isFixed="false"/>
<part id="5" parentId="4" blockId="Link2" parentAssemblyJointIndex="0" assemblyJointIndex="0" parentAxis1="-x" childAxis1="x"
                          parentAxis2="-x" childAxis2="x" rotationAxisZ="y" rotationAxisX="x" jointOption="0" isFixed="false"/>
```

Fig. 9. Assembly data file

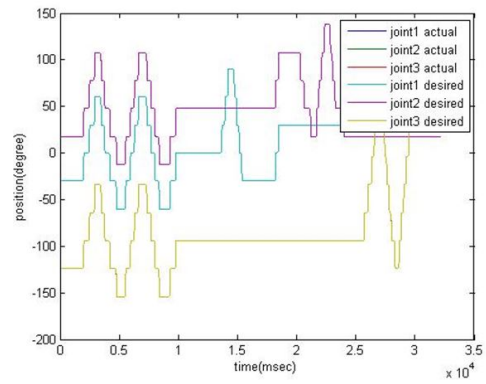

Fig. 11. Position trajectory of each joint

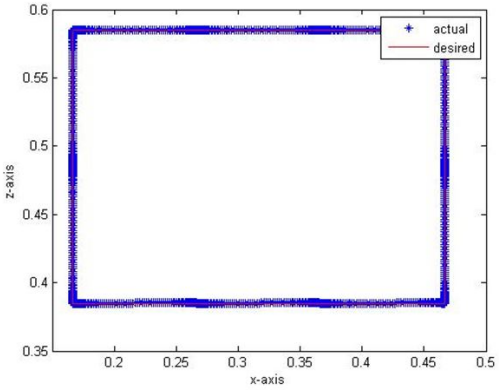

Fig. 12. Position trajectory (task motion) in x-z plane

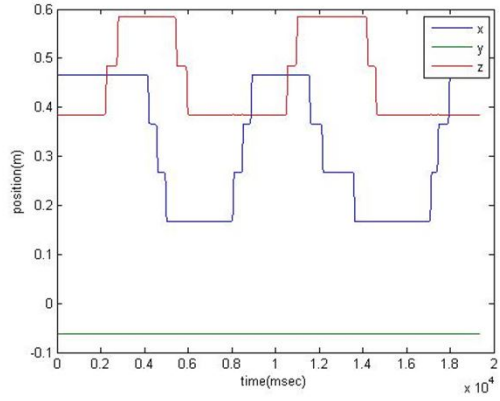

Fig. 13. Position trajectory in x, y, z axis

시뮬레이션 환경은 동역학 시뮬레이션 엔진을 포함하고 있다. 시뮬레이터에 구현된 모듈라 로봇을 Fig. 10에 제시하였다. 구현된 로봇 모델은 CAD 데이터를 기반으로 동역학 파라미터가 반영되어 있다. 구현된 모듈라 로봇에 대하여 조인트 공간과 작업 공간 상에서 모션 제어 시뮬레이션을 수행하였다. 우선 조인트별 기준 궤적을 각 축별로 30도씩 증감시키면서 위치 변화를 하도록 하여 각 조인트의 추종성능을 검증하였다. 각 조인트의 목표치가 point-to-point로 주어지므로 5차 보간 알고리즘을 사용하여 기준궤적을 생성하였다. Fig. 11은 각 조인트별 기준 궤적 및 실제 위치값을 나타내고 있다. 구성된 모듈라 로봇이 기준 궤적대로 잘 움직이는 것을 확인할 수 있다. 다음으로 작업공간상에서의 모션 시뮬레이션을 수행하였다. 자동 생성된 기구학 정보를 활용하여 역기구학을 풀어 작업 공간상에서의 제어가 이루어지고 있다. 3자유도 매니퓰레이터이므로 실제 task space상에서의 움직임은 제한이 있으므로 x-z 평면상에서 위치 변화를 하도록 하였다. Fig. 12는 x-z 평면상의 위치 궤적을 나타내고 있고, Fig. 13은 그 때의 x, y, z 좌표상의 기준 궤적 및 실제 위치를 보여주고 있다. 시뮬레이션 결과에서 기준 궤적을 잘 추종하고 있다. 즉, 자동생성된 기구학 및 제어 엔진을 적용하여 구성된 3자유도 모듈라 로봇의 제어가 잘 이루어지고 있음을 확인할 수 있다.

4. 결 론

본 논문은 자가 재구성이 가능한 모듈라 로봇의 작업 적응성 향상을 위하여 기구학/동역학 연산을 포함하는 제어 엔진의 자가 재구성 방법을 제안하였다. 사용자가 재구성한 모듈라 로봇 하드웨어로부터 연결정보를 추출하여 기구학/동역학 정보를 추출하고 로봇 제어에 필요한 연산코드를 자동적으로 생성하였다. 로봇의 자가재구성부터 모션제어까지의 과정을 시뮬레이션을 통하여 검증함으로써 제안한 방법의 효용성을 입증하였다.

References

[1] K. Gilpin and D. RUS, “Modular Robot Systems,” IEEE Robotics & Automation Magazine, vol.17, no.3, pp.38-55, 2010.

[2] M. Yim, Y. Zhang, and D. Duff, “Modular Robots,” IEEE Spectrum, vol.39, no.2, pp.30-34, 2002.

[3] G. Liu, Y. Liu, and A.A. Goldenberg, “Design, Analysis, and Control of a Spring-Assisted Modular and Reconfigurable

Robot," IEEE Trans. on Mechatronics, vol.16, no.4, 2011.
[4] S. Castro, S. Koehler, and H. Kress-Gazit, "High-Level Control of Modular Robots," IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011.
[5] W-H. Zhu, T. Lamarche, E. Dupuis, D. Jameux, P. Barnard, and G. Liu, "Precision Control of Modular Robot Manipulators: The VDC Approach With Embedded FPGA," IEEE Trans. on Robotics, vol.29, no.5, 2013.
[6] S. Hong, W. Lee, H. Lee, and S. Kang, "Joint and Link Module Geometric Shapes of Modular Manipulator for Various Joint Configurations," Journal of Korea Robotics Society, vol.11, no.3, 2016.
[7] G. Liu, S. Abdul, and A.A. Goldenberg, "Distributed Modular and Reconfigurable Robot Control with Torque Sensing," IEEE International Conference on Mechatronics and Automation, 2006.
[8] K.J. Åström, and T. Hägglund, Automatic Tuning of PID Controllers, Instrument Society of America, 1988.

도 현 민

1997 서울대학교 전기공학부(공학사)
1999 서울대학교 전기공학부(공학석사)
2004 서울대학교 전기컴퓨터공학부(공학박사)
2010~현재 한국기계연구원 로봇메카트로닉스연구실

관심분야: 산업용 로봇 제어, 로봇 안전기술

최 태 용

2003 POSTECH 전자전기공학과(공학사)
2010 KAIST 전기전자공학과(공학박사)
2011~현재 한국기계연구원 로봇메카트로닉스연구실

관심분야: 산업용 로봇 제어, 산업 지능

박 동 일

2000 KAIST 기계공학과(공학사)
2002 KAIST 기계공학과(공학석사)
2006 KAIST 기계공학과(공학박사)
2006~현재 한국기계연구원 로봇메카트로닉스연구실

관심분야: 로봇 해석 및 설계, 로봇 머니퓰레이터

김 두 형

1982 서울대학교 공과대학(공학사)
1990 한국과학기술원 기계공학과(공학석사)
2003 한국과학기술원 기계공학과(공학박사)
1982~현재 한국기계연구원 로봇메카트로닉스연구실

관심분야: 산업용 로봇 설계, 자동화 기술

손 영 수

1981 동국대학교 전자공학과(공학사)
1984 동국대학교 전자공학과(공학석사)
1997 동국대학교 전자공학과(공학박사)
1989~현재 한국기계연구원 로봇메카트로닉스연구실

관심분야: 자동화 시스템 제어, 반도체 공정장비