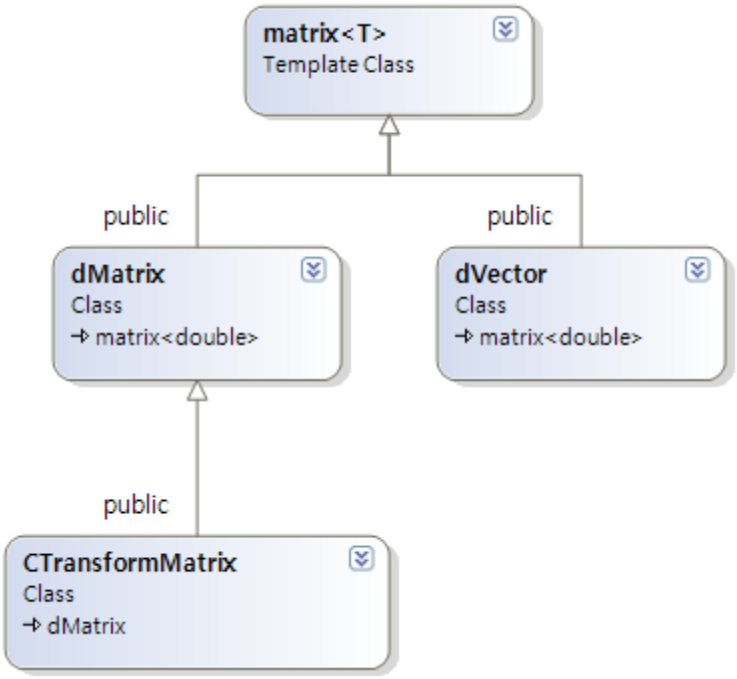
Robotics API 설명

KITECH 양광웅 작성
Page365@gmail.com

행렬과 벡터

행렬과 벡터는 메니퓰레이터의 기구학과 동역학을 계산하는데 필수적으로 사용되는 자료형이다. 행렬의 연산에는 Techsoft에서 개발한 Matrix TCL Lite를 사용한다.

Techsoft website URL : http://www.techsoftpl.com/matrix/index.htm



matrix<T> 클래스는 행렬 연산을 위한 각종 기능을 제공한다. 이 클래스를 상속 받아 행렬을 표현하기 위한 dMatrix와 벡터를 표현하기 위한 dVector 클래스를 만든다. CTransformMatrix 는 메니퓰레이터의 위치 정보와 자세 정보를 동시에 표현하는 동차 변환(homogeneous transformation) 행렬이다.

* 01. Transformation.docx 문서 참조

기구학

기구학은 로봇 매니퓰레이터를 이루는 링크와 관절의 움직임을 다루며, 힘과 가속도 같은 역학적 요소를 다루는 동역학과는 구별된다.

각 관절 값들과 말단장치의 위치 및 방향 관계식을 기구학 이론을 사용하여 풀 수 있으며 해석을 위해서는 다음과 같은 가정이 필요하다.

1. 매니퓰레이터는 강체로 가정한다.
2. 매니퓰레이터는 충분히 느리게 움직여 관성을 무시 할 수 있다.

기구학은 다음과 같이 정기구학(forward kinematics)과 역기구학(inverse kinematics)으로 분류할 수 있다.

- 정기구학: 매니퓰레이터의 각 관절의 각도가 주어진 경우, 말단장치의 위치와 방향을 계산하는 것이며, 유일해가 존재한다.
- 역기구학: 말단장치의 위치와 방향을 알 때, 각 관절의 각도값을 계산하는 것이며, 해가 여러개 존재 할 수 있다.

CKinematics 클래스

CKinematics 클래스는 기구학을 풀기위한 기본 기능을 제공한다.

dVector GetDesired ();

메니퓰레이터 말단장치 혹은 무게중심이 도달해야 할 목적 위치를 벡터로 반환한다.

리턴 값:

- 위치인 경우: $\begin{bmatrix} x & y & z \end{bmatrix}^T$
- 위치와 자세인 경우: $\begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^T$

dVector GetCurrent ();

메니퓰레이터 말단장치 혹은 무게중심이 위치한 현재 위치를 벡터로 반환한다.

리턴 값:

- 위치인 경우: $\begin{bmatrix} x & y & z \end{bmatrix}^T$
- 위치와 자세인 경우: $\begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^T$

void AttachJoint (eJointType type, int axis, double x, double y, double z, double phi, double theta, double psi, double radius, double limit_lo, double limit_hi, double median, double weight);

메니퓰레이터를 구성하는 관절-링크 쌍을 목록에 추가 한다.

eJointType 형식의 type은 다음과 같이 3종류를 가진다.

- FIXED_JOINT - 고정된관절: 좌표를 이동하기위해 사용한다.
- REVOLUTE_JOINT - 회전운동관절: 회전 축을 중심으로 회전운동 한다.
- PRISMATIC_JOINT - 직선운동관절: 직선 축을 따라 직선이동 한다.

axis는 다음과 같이 3종류를 설정한다.

- 0 - x 축을 따라 회전하거나 직선이동 한다.
- 1 - y 축을 따라 회전하거나 직선이동 한다.
- 2 - z 축을 따라 회전하거나 직선이동 한다.

x, y, z, phi, theta, psi는 이전 관절의 좌표를 기준으로 현재 관절의 좌표를 변환한다.
* 01. Transformation.docx 문서 참조

radius는 관절과 관절을 연결하는 링크의 반지름이다.

limit_lo와 _limit_hi는 관절의 구동 범위 (limit_lo ~ limit_hi) 이다. 만일 limit_hi가 limit_lo보다 작으면 관절의 구동 범위는 관절 값에 영향을 주지 않는다.

median은 관절의 중간 값으로 Null motion을 만들어내기 위해 사용된다. 역기구학을 풀 때, 관절은 여유자유도를 이용하여 median 근처 값으로 수렴하려 한다.

weight는 관절의 가중치로 다른 관절에 비해 weight값이 클수록 다른 관절에 비해 잘 움직인다.

dVector SolveJTR (double minAlpha, double clampMag);
SolveJTR() 함수는 Jacobian Transpose 방법으로 역기구학을 풀어, 메니퓰레이터 말단 장치 혹은 무게중심이 추종해야 할 목적 위치로 이동하기 위한 관절들의 변화량을 계산하여 벡터로 리턴한다.

minAlpha는 Buss & Kim이 제안한 Jacobian Transpose 방법으로 계산된 이득(gain)의 최소 값이다.

clampMag는 메니퓰레이터가 도달해야 할 목표 위치가 최대값 (clampMag로 지정된 값) 보다 멀 경우, 이를 최대값 안으로 끌어당겨 준다.

리턴 값: $\begin{bmatrix} \Delta q_1 & \Delta q_2 & \cdots & \Delta q_n \end{bmatrix}^T$

* 04. Inverse Kinematics.docx 문서 참조

dVector SolveDLS (double dlsLambda, double clampTask, double clampNull);
SolveDLS() 함수는 Damped least square 방법으로 역기구학을 풀어, 메니퓰레이터 말단 장치 혹은 무게중심이 추종해야 할 목적 위치로 이동하기 위한 관절들의 변화량을 계산하여 벡터로 리턴한다.

dlsLambda는 Damped least square 방법으로 역기구학을 풀 때 사용되는 댐핑 상수다. 댐핑 상수가 클수록 메니퓰레이터의 움직임이 둔감해지고 말단 장치에서의 위치 오차도 커진다. 반면 댐핑 상수가 작을수록 메니퓰레이터의 움직임은 빠르고 특이점 근처에서 진동하며 불안해 진다.

clampTask는 메니퓰레이터가 도달해야 할 목표 위치가 최대값 (clampTask로 지정된 값) 보다 멀 경우, 이를 최대값 안으로 끌어당겨 준다.

ClampNull은 영공간(null space)에 할당한 작업(현재 관절 각을 median으로 수렴시키는)의 최대값을 clampNull에서 설정한 값보다 커지지 않도록 한다.

리턴 값: $\begin{bmatrix}\Delta q_1 & \Delta q_2 & \cdots & \Delta q_n\end{bmatrix}^T$

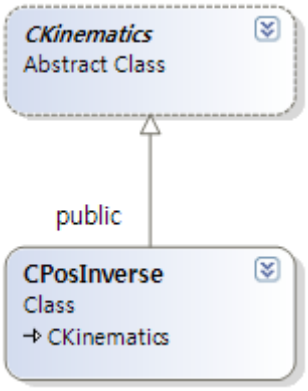
* 04. Inverse Kinematics.docx 문서 참조

dVector GetJointAngle ();
메니퓰레이터를 구성하는 모든 관절의 각도를 벡터로 반환한다.

리턴 값: $\begin{bmatrix}q_1 & q_2 & \cdots & q_n\end{bmatrix}^T$

void Apply (dVector &dq);
SolveJTR()이나 SolveDLS() 함수는 메니퓰레이터를 구성하는 모든 관절들의 변화량을 계산하여 리턴한다. 이 함수들이 리턴한 값(dq)을 Apply() 함수로 현재 관절 값에 더함으로 메니퓰레이터를 새로운 자세로 움직인다.

CPosInverse 클래스
CPosInverse클래스는 공간상에서 메니퓰레이터 말단장치가 추종해야 위치가 주어졌을 때 관절의 변화량을 계산하기 위한 클래스다. 이 클래스는 CKinematics 클래스를 상속함으로 기구학을 풀기 위한 기본 기능을 CKinematics 클래스에 의존한다.

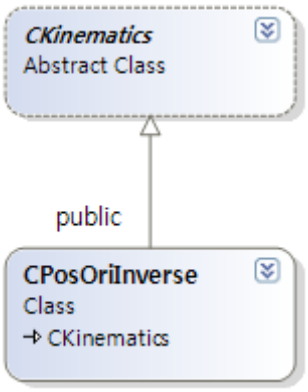


* 05. Jacobian.docx 문서 참조

void SetDesired (double x, double y, double z);
메니퓰레이터 말단장치가 추종해야 할 위치 값 (x, y, z) 을 설정한다.

CPosOriInverse 클래스

CPosInverse클래스는 공간상에서 메니퓰레이터 말단장치가 추종해야 위치와 자세가 주어졌을 때 관절의 변화량을 계산하기 위한 클래스다. 이 클래스는 CKinematics 클래스를 상속함으로 기구학을 풀기위한 기본 기능을 CKinematics 클래스에 의존한다.



* 05a. Analytic Jacobian.docx 문서 참조
* 05b. Inverse with Quaternion.docx 문서 참조

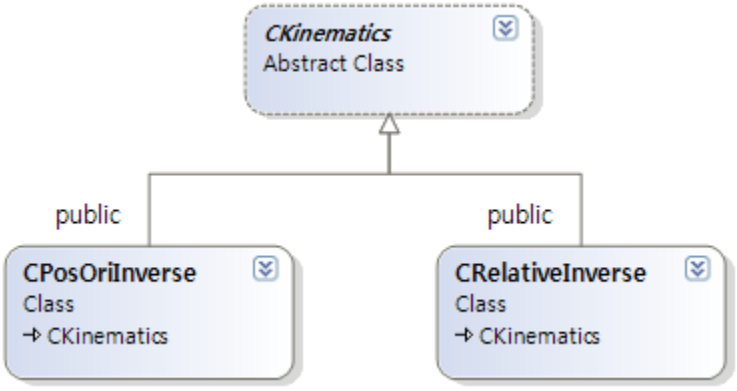
CPosOriInverse (int inverseAxis);
CPosOriInverse 클래스를 생성할 때, 역기구학을 푸는 옵션을 inverseAxis 에서 설정한다.
inverseAxis 는 다음의 3 가지 값 중 하나를 가진다.

- POSITION_ONLY - 자세는 무시하고 위치에 대한 역기구학을 푼다.
- ORIENTATION_ONLY - 위치는 무시하고 자세에 대한 역기구학을 푼다.
- POSITION_ORIENTATION - 위치와 자세에 대한 역기구학을 푼다.

void SetDesired (double x, double y, double z, double phi, double theta, double psi);
메니퓰레이터 말단장치가 추종해야 할 위치와 자세 값 $(x, y, z, \phi, \theta, \psi)$을 설정한다.

CRelativeInverse 클래스

CRelativeInverse 클래스는 양팔 로봇이 왼 손으로 물건을 쥐고 오른 손으로 작업을 하는 경우와 같이 왼 손을 원점으로 오른 손의 작업을 지시할 때 사용한다.



* 06. Relative Jacobian.docx 문서 참조

CRelativeInverse(CKinematics *kinL, CKinematics *kinR);
CRelativeInverse 클래스는 하나의 메니퓰레이터 말단장치의 좌표를 기준으로 다른 하나의 메니퓰레이터 말단장치의 작업 위치를 계산하기 때문에, 먼저 작성한 CPosOriInverse 클래스로 두 개의 메니퓰레이터에 대한 인스턴스를 만든 후, 이 두 포인터를 CRelativeInverse 클래스의 생성자의 인자로 넘겨준다.

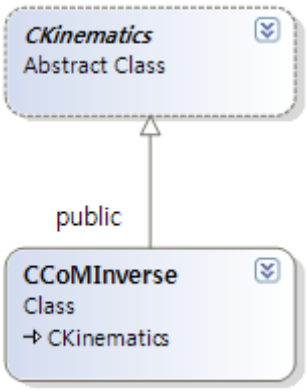
```
CPosOriInverse   *_invL = new CPosOriInverse (POSITION_ORIENTATION);
_invL->AttachJoint (...);
_invL->AttachJoint (...);

CPosOriInverse   *_invR = new CPosOriInverse (POSITION_ORIENTATION);
_invR->AttachJoint (...);
_invR->AttachJoint (...);

CRelativeInverse   *_rel = new CRelativeInverse (_invL, _invR);
```

void Apply (dVector &dq);

Apply() 함수는 CKinematics 클래스에서 제공하는 기능이나, CRelativeInverse 클래스에서는 관절의 변화량(dq)을 두 메니퓰레이터에 동시에 적용하여야 함으로 오버라이딩 하였다. CKinematics 클래스에서 제공하는 기능과 같이, SolveJTR()이나 SolveDLS() 함수들이 리턴한 값(dq)을 Apply() 함수로 현재 관절 값에 더함으로 메니퓰레이터를 새로운 자세로 움직인다.

CCoMInverse 클래스

CCoMInverse 클래스는 무게중심(CoM; Center of Mass)의 위치를 원하는 지지 영역 위에 유지하기 위한 역기구학을 푼다.



* 07. CoM Jacobian.docx 문서 참조

CCoMInverse (int comAxis);

CCoMInverse 클래스를 생성할 때, 역기구학을 푸는 옵션을 comAxis 에서 설정한다. comAxis 는 다음의 3가지 값의 조합으로 만들어진다.

- COM_AXIS_X - 무게 중심의 목적 위치 중 x 축을 활성화 한다.
- COM_AXIS_Y - 무게 중심의 목적 위치 중 y 축을 활성화 한다.
- COM_AXIS_Z - 무게 중심의 목적 위치 중 z 축을 활성화 한다.

예를 들자면, 만일 로봇이 넘어지지 않도록 x, y 평면 위에 원하는 지지 영역을 만들고 로봇의 무게 중심이 이 영역 안에서 유지되도록 하기 위해서는 comAxis 값을 COM_AXIS_X|COM_AXIS_Y 로 설정할 수 있다.

void AttachJoint (eJointType type, int axis, double mass, double cx, double cy, double cz, double x, double y, double z, double phi, double theta, double psi, double radius);

메니퓰레이터를 구성하는 관절-링크 쌍을 목록에 추가 한다. CKinematics 클래스의 AttachJoint () 함수와 동일한 기능을 수행하며 추가된 부분은 다음과 같다.

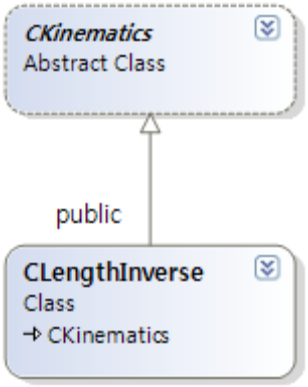
mass는 링크의 질량이다. (단위: Kg)

cx, cy, cz는 이전 조인트의 좌표계를 기준으로 한 현재 링크의 무게중심 위치다.

void SetDesired (double x, double y, double z);
메니퓰레이터의 무게중심이 추종해야 할 위치 값 (x, y, z) 을 설정한다.

CLengthInverse 클래스

로봇이 양손으로 물건을 잡는 경우를 생각해 볼 때, 양 손간의 거리는 항상 일정하게 유지된다. 이 때 양 손간의 거리를 일정하게 유지하기 위하여 CLengthInverse 클래스를 사용할 수 있다. CLengthInverse 클래스는 메니퓰레이터의 말단장치가 공간상의 한 점 (x, y, z) 을 중심으로 반지름 (length)을 가지는 구의 표면을 추종하도록 역기구학을 푼다.



* 08. Length Jacobian.docx 문서 참조

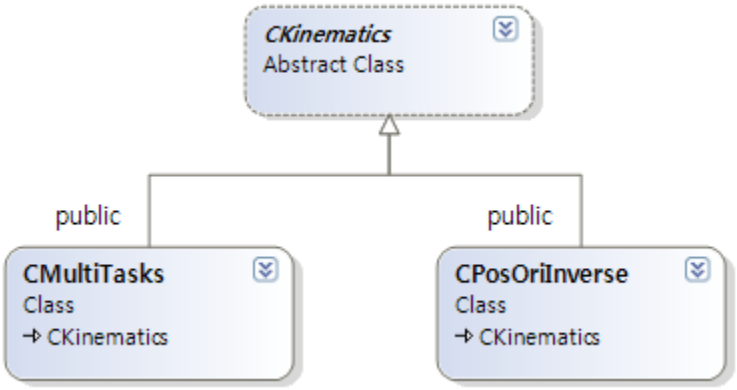
void SetDesired (double x, double y, double z, double length);
메니퓰레이터의 말단장치가 추종해야 할 위치 값 (x, y, z) 과 위치 값으로부터 떨어진 거리 (length)을 설정한다.

여유자유도 기구학

여유자유도는 주어진 작업 공간 자유도 이상의 관절 공간 자유도를 보유한 경우를 의미하며, 이 경우 주어진 작업을 수행하고 남는 자유도를 활용하여 추가적인 작업이 가능하다.

CMultiTasks 클래스

CMultiTasks 클래스는 두 개 이상의 작업이 주어졌을 때, 작업의 우선순위에 따라 역기구학을 푼다. 이 때 메니퓰레이터는 각각의 주어진 작업 공간 자유도 이상의 관절공간 자유도를 필요로 하며, 우선 순위가 높은 작업을 수행하고 남는 자유도를 활용하여 순위가 낮은 작업을 순차적으로 수행한다.



* 11. 병렬 작업.docx 문서 참조
* 12. 우선순위 작업.docx 문서 참조

여기서 CKinematics 클래스의 SolveJTR () 함수나 SolveDLS () 함수를 사용하여 역기구학을 풀 경우에는 CMultiTasks 클래스에 등록된 작업의 우선 순위를 고려하지 않고 동일한 순위로 풀린다.

void AddTask (CKinematics *task);

한 개 이상의 작업에 대한 클래스 인스턴스를 만든 후, CMultiTasks 클래스에 AddTask () 함수로 순차적으로 등록한다. 작업을 등록할 때 처음에 추가한 작업이 우선순위가 낮다.

```
CObjectContainer *_objs = new CObjectContainer ();
_objs->Push ();
_objs->AddObject (new ObjectJoint (...));
_objs->AddObject (new ObjectJoint (...));
...
_objs->Pop ();

CPosOriInverse *_inv1 = new CPosOriInverse (POSITION_ONLY);
_inv1->AttachJoint (_objs->FindObject ("link1"));
_inv1->AttachJoint (_objs->FindObject ("link2"));
...

CPosOriInverse *_inv2 = new CPosOriInverse (POSITION_ONLY);
_inv2->AttachJoint (_objs->FindObject ("link1"));
_inv2->AttachJoint (_objs->FindObject ("link2"));

CMultiTasks *_tasks = new CMultiTasks ();
_tasks->AddTask (_inv1);
_tasks->AddTask (_inv2);
```

dVector SolveChi (double dlsLambda, double clampTask);

Chiaverini가 제안한 방법으로 작업의 우선 순위에 따라 역기구학을 풀고 관절들의 변화량을 계산하여 벡터로 리턴한다.

우선 순위 작업의 Null-Space에 근거한 기본 이론에 따라서 유도된 방식이나, 앞선 우선순위 작업과 현재 작업의 동시 작업이 불가능한 경우 Algorithmic 특이점이 발생한다.

dlsLambda는 Damped least square 방법으로 역기구학을 풀 때 사용되는 댐핑 상수다.

clampTask는 메니퓰레이터가 도달해야 할 목표 위치가 최대값 (clampTask로 지정된 값) 보다 멀 경우, 이를 최대값 안으로 끌어당겨 준다.

리턴 값: $\left[\Delta q_1 \quad \Delta q_2 \quad \cdots \quad \Delta q_n\right]^T$

dVector SolveNak (double dlsLambda, double clampTask);

Nakamura가 제안한 방법으로 작업의 우선 순위에 따라 역기구학을 풀고 관절들의 변화량을 계산하여 벡터로 리턴한다.

Algorithmic 특이점일 회피하기 위해서 새롭게 제시된 방법으로, 특이점은 발생하지 않으나, 낮은 우선 순위 작업의 경우 많은 왜곡이 발생하여 추종 Error가 크다.

dlsLambda는 Damped least square 방법으로 역기구학을 풀 때 사용되는 댐핑 상수다.

clampTask는 메니퓰레이터가 도달해야 할 목표 위치가 최대값 (clampTask로 지정된 값) 보다 멀 경우, 이를 최대값 안으로 끌어당겨 준다.

리턴 값: $\begin{bmatrix} \Delta q_1 & \Delta q_2 & \cdots & \Delta q_n \end{bmatrix}^T$

동역학

동역학은 힘이 로봇 매니퓰레이터의 운동에 미치는 영향을 다룬다.
매니퓰레이터의 동역학적 모델을 유도하는 것은 동작의 시뮬레이션(simulation)과 매니퓰레이터의 구조 해석, 컨트롤 알고리즘의 설계에 중요한 역할을 한다. 시뮬레이션은 실제 로봇을 사용하지 않고 제어 알고리즘과 동작 계획을 테스트 할 수 있도록 한다.

CDynamics 클래스

CDynamics 클래스는 동역학을 풀기위한 기본 기능을 제공한다.

void AttachJoint (eJointType type, int axis, double mass, double cx, double cy, double cz, double Ixx, double Iyy, double Izz, double Ixy, double Iyz, double Izx, double x, double y, double z, double phi, double theta, double psi, double radius);

메니퓰레이터를 구성하는 관절-링크 쌍을 목록에 추가 한다.

eJointType 형식의 type은 다음과 같이 3종류를 가진다.

- FIXED_JOINT - 고정된관절: 좌표를 이동하기위해 사용한다.
- REVOLUTE_JOINT - 회전운동관절: 회전 축을 중심으로 회전운동 한다.
- PRISMATIC_JOINT - 직선운동관절: 직선 축을 따라 직선이동 한다.

axis는 다음과 같이 3종류를 설정한다.

- 0 - x 축을 따라 회전하거나 직선이동 한다.
- 1 - y 축을 따라 회전하거나 직선이동 한다.
- 2 - z 축을 따라 회전하거나 직선이동 한다.

mass는 링크의 질량이다. (단위: Kg)

cx, cy, cz는 이전 조인트의 좌표계를 기준으로 한 현재 링크의 무게중심 위치다.

Ixx, Iyy, Izz, Ixy, Iyz, Izx는 링크의 관성 텐서 행렬의 원소다. 관성 텐서 행렬은 다음과 같이 구성된다.

$$\mathbf{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{zx} \\ I_{xy} & I_{yy} & I_{yz} \\ I_{zx} & I_{yz} & I_{zz} \end{bmatrix}$$

x, y, z, phi, theta, psi는 이전 관절의 좌표를 기준으로 현재 관절의 좌표를 변환한다.
* 01. Transformation.docx 문서 참조

radius는 관절과 관절을 연결하는 링크의 반지름이다.

int GetJointNo ();
관절의 개수를 리턴한다.

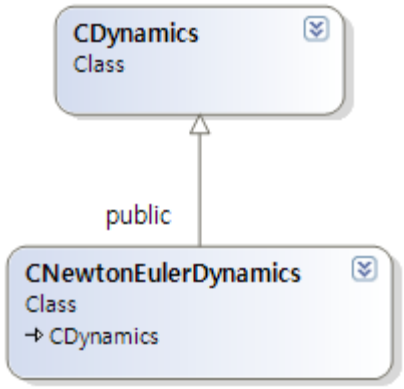
vector<JointInfo *> &GetJointList ();
관절-링크 쌍을 담고 있는 벡터를 리던한다.

dVector GetJointAngle ();
메니퓰레이터를 구성하는 모든 관절의 각도를 벡터로 반환한다.

리턴 값: $\begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}^T$

CNewtonEulerDynamics 클래스

뉴턴 오일러 공식을 이용하면, 운동방정식은 두 단계의 반복적인 절차로 구할 수 있다. 베이스에서 말단장치까지의 속도와 가속도의 전파를 위한 순회귀(forward recursion)와 말단장치에서 베이스까지의 힘과 모멘트의 전파를 위한 역회귀(backward recursion)이 그것이다.
CNewtonEulerDynamics 클래스는 뉴턴 오일러 공식을 이용하여 정/역 동역학을 계산한다.



* 21. Newton-Euler Formulation.docx 문서 참조

CNewtonEulerDynamics(double gravity[3], double fsi, double fvi);
CNewtonEulerDynamics 클래스의 생성자에서 중력과 마찰력을 설정한다.

gravity는 공간상에서 중력 벡터다. 아래와 같이 지표면에서의 수직 방향 중력 9.81을 설정한다.

```
double gravity[3] = {0., 0., -9.81 };
```

fsi와 fvi는 마찰 계수로 다음과 같다.

- fsi - coulomb friction coefficient
- fvi - viscous friction coefficient

dVector ForwardDynamicsRK4 (dVector &torque, double dt);
메니퓰레이터의 관절에 토크 (torque)가 주어졌을 때, 4차 런지쿠타(Fourth-order Runge-Kutta) 법으로 관절의 가속도를 계산한다.

torque는 각 관절에 가하는 토크다. 즉 메니퓰레이터를 제어하기 위해 관절에 붙은 모터가 관절에 가하는 힘이다.

dt는 시간 간격이다.

리턴 값: $\begin{bmatrix}\Delta\dot{q}_1 & \Delta\dot{q}_2 & \cdots & \Delta\dot{q}_n\end{bmatrix}^T$

dVector ForwardDynamics (dVector &torque);
메니퓰레이터의 관절에 토크 (torque)가 주어졌을 때, 관절의 가속도를 계산한다.

torque는 각 관절에 가하는 토크다. 즉 메니퓰레이터를 제어하기 위해 관절에 붙은 모터가 관절에 가하는 힘이다.

리턴 값: $\begin{bmatrix}\Delta\dot{q}_1 & \Delta\dot{q}_2 & \cdots & \Delta\dot{q}_n\end{bmatrix}^T$

dVector InverseDynamics (dVector &accel);
메니퓰레이터의 관절에 가속도 (accel)가 주어졌을 때, 관절에 작용하는 토크를 계산한다.

ddq는 각 관절에 가하는 가속도다.

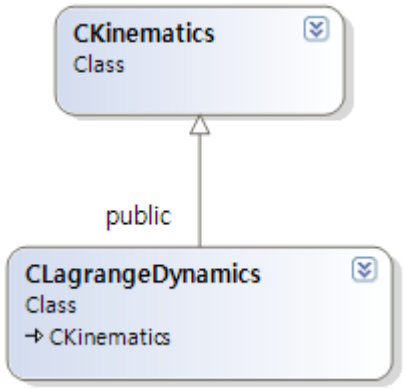
리턴 값: $\begin{bmatrix}\tau_1 & \tau_2 & \cdots & \tau_n\end{bmatrix}^T$

void Apply (dVector &accel, double dt);
관절에 작용하는 가속도 값 (accel)을 Apply() 함수로 현재 관절 속도 값에 더함으로 메니퓰레이터를 새로운 속도와 자세로 움직인다.

void Reset ();
모든 관절의 위치, 속도, 가속도 값을 0으로 초기화 한다.

CLagrangeDynamics 클래스

CLagrangeDynamics 클래스는 Euler-Lagrange 운동 방정식을 이용하여 정/역 동역학을 계산한다.



* 22. Differentiation of Jacobian.docx 문서 참조
* 23 Lagrangian Formulation.docx 문서 참조

CLagrangeDynamics (double gravity[3], double fsi, double fvi);

CLagrangeDynamics 클래스의 생성자에서 중력과 마찰력을 설정한다.

gravity는 공간상에서 중력 벡터다. 아래와 같이 지표면에서의 수직 방향 중력 9.81을 설정한다.

```
double gravity[3] = {0., 0., -9.81 };
```

fsi와 fvi는 마찰 계수로 다음과 같다.

- fsi - coulomb friction coefficient
- fvi - viscous friction coefficient

dVector ForwardDynamics (dVector &torque);

메니퓰레이터의 관절에 토크 (torque)가 주어졌을 때, 관절의 가속도를 계산한다.

torque는 각 관절에 가하는 토크다. 즉 메니퓰레이터를 제어하기 위해 관절에 붙은 모터가 관절에 가하는 힘이다.

리턴 값: $\begin{bmatrix} \Delta\dot{q}_1 & \Delta\dot{q}_2 & \cdots & \Delta\dot{q}_n \end{bmatrix}^T$

dVector InverseDynamics (dVector &accel);

메니퓰레이터의 관절에 가속도 (accel)가 주어졌을 때, 관절에 작용하는 토크를 계산한다.

ddq는 각 관절에 가하는 가속도다.

리턴 값: $\begin{bmatrix} \tau_1 & \tau_2 & \cdots & \tau_n \end{bmatrix}^T$

void Apply (dVector &accel, double dt);
관절에 작용하는 가속도 값 (accel)을 Apply() 함수로 현재 관절 속도 값에 더함으로 메니퓰레이터를 새로운 속도와 자세로 움직인다.

void Reset ();
모든 관절의 위치, 속도, 가속도 값을 0으로 초기화 한다.

double GetKineticEnergy();
전체 링크의 에너지는 운동 에너지와 위치 에너지의 합이다. GetKineticEnergy() 함수는 전체 링크의 운동 에니지를 리턴한다.

double GetPotentialEnergy();
GetPotentialEnergy () 함수는 전체 링크의 위치 에니지를 리턴한다.

경로 보간

로봇이 작업하면서 거쳐가야 할 몇 개의 불연속 점이 주어졌을 때, 로봇은 연속적으로 점과 점 사이를 움직여야 한다. 이 때 주어진 불연속 점들에 대하여 보간법으로 중간점들을 구하고, 로봇은 이 점들을 연속하여 지나감으로 작업을 부드럽게 마무리 할 수 있다.

여기서 사용 가능한 보간법은 다음과 같다.

- Linear interpolation
- Cardinal spline
- Monotone cubic interpolation
- Cubic spline interpolation

* 30. Interpolation.docx 문서 참조

Linear interpolation

가장 단순한 보간법으로 두 점과 점 사이의 점을 1차 다항식으로 계산하는 방식이다.

```
template <typename T>
vector<T> LinearInterpolation (vector<T> &x, double timeSlice);
```

x는 이산 점들의 배열이다.
timeSlice는 보간 점들이 만들어질 시간 간격이다. (단위: sec.)

리턴 값: 보간 점들의 배열

위 함수를 호출하기 위해 점들의 배열은 다음의 규칙에 따라 저장되어야 한다.

$$\begin{bmatrix}\mathbf{t} & \mathbf{p}\end{bmatrix} = \begin{bmatrix} t_1 & p_{11} & p_{12} & \cdots & p_{1m} \\ t_2 & p_{21} & p_{22} & \cdots & p_{2m} \\ t_3 & p_{31} & p_{32} & \cdots & p_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_n & p_{n1} & p_{n2} & p_{n3} & p_{nm} \end{bmatrix}$$

여기서 $\begin{bmatrix} t_i & p_{i1} & p_{i2} & \cdots & p_{im} \end{bmatrix}$는 t_i 시간에 로봇이 위치해야 할 점 p_i를 나타낸다.

점들의 배열은 다음 예에서와 같이 만들어 질 수 있다.

```
double a[][4] = {
        // time,    x,      y,      z
        { 0.,       0.,     0.,     0.5, },
        { 1.,       1.,     0.,     0.5, },
        { 2.,       1.,     1.,     0.5, },
        { 3.,       0.,     1.,     0.5, },
        { 4.,       0.,     0.,     0.5, },
        { 5.,       0.,     0.,     1., },
        { 6.,       1.,     0.,     1., },
        { 7.,       1.,     1.,     1., },
        { 8.,       0.,     1.,     1., },
        { 9.,       0.,     0.,     1., },
};

vector<valarray<double>> _node;

for (unsigned int i=0; i<sizeof(a)/32; ++i) {
        _node.push_back (valarray<double>(&a[i][0], 4));
}
```

Cardinal spline

Cardinal spline은 Cubic Hermite spline의 일종으로 Hermite 형식을 만족하는 3차 spline 곡선이다. Tension 파라메터를 바꿈으로 곡선의 모양을 적절히 선택할 수 있다.

```
template <typename T>
vector<T> CardinalSplineInterpolation (vector<T> &x, double timeSlice, double tension);
```

tension이 곡선의 모양을 결정한다.

Monotone cubic interpolation

Monotone cubic 보간은 Cubic Hermite spline의 일종으로 Hermite 형식을 만족하는 3차 spline 곡선이다. 기울기(m_k)에서 오버슈트를 억제하여 각 점들이 단조롭게 이어지도록 한다.

```
template <typename T>
vector<T> MonotoneCubicInterpolation (vector<T> &x, double timeSlice, double monotonicity);
```

Cubic spline interpolation

Cubic spline 보간은 각 점을 지나는 곡선에서 위치와 속도, 가속도의 연속성을 보장하므로 로봇

의 동작을 부드럽게 제어할 수 있다.

```
template <typename T>
vector<T> CubicSplineInterpolation (vector<T> &x, double timeSlice);
```