보행 보고서

19기 예비단원 이원준

1. 보행 이론 기본 개념 정리

- COM (center of mass): 질량 중심점으로 로봇의 전체 무게가 집중된 가상의 지점을 의미한다. 로봇이 균형을 유지하기 위해 중심을 맞춰야하는 지점이다. 일반적으로 안정적인 보행을 하기 위해 지지다각형을 만들기 위해서는 바닥면에 선형 투사한 점을 사용한다.

- COP (center of pressure): 압력 중심점으로 로봇의 발이 지면과 접촉하는 면적 중 실제로 힘이 작용하는 지점이다. COP는 지면과의 접촉면에서 생성되는 반발력이 결정하고 발바닥 전체에 걸친 압력 분포의 중심을 나타낸다.

- 지지 다각형 : 로봇이 지면에 닿고 있는 발의 각 지점을 연결하여 형성되는 다각형이다. 바닥면에 2차원으로 형성되는 안전한 영역이라고 이해할 수 있다. 로봇이 SS일때는 해당 발바닥이, DS일때는 두 발바닥이 포함된다. COP와 COM을 기반으로 지지다각형을 만들 수 있다. 지지 다각형은 COP와 COM 수직 투영점을 연결한 직선의 주위 공간에 형성된다.

- ZMP(Zero moment point) : 로봇이 보행하는 동안 지면에 접촉하는 부분에서 수직으로 작용하는 힘과 모멘트의 합이 0이 되는 지점이다. 이 지점이 지지 다각형 내부에 존재하면 로봇이 안정적인 보행을 수행할 수 있다. 로봇이 SSP일때와 DSP일때 ZMP의 경로가 서로 다르게 정의된다.

- EE(end effector): manipulator(link와 joint의 결합체)의 끝단을 말한다.

- LIPM(Liner Inversed Pendulum Model): 선형 역진자 모델, 로봇의 움직임을 단순화 하여 무게중심의 움직임을 예측하고 제어하는데 활용한다. 이 모델은 로봇이 균형을 유지하면 자연스럽게 보행할 수 있도록 도와주는 수학적 도구이다.

이 모델은 로봇의 보행을 이해하기 위해 몇가지 가정을 한다.

1. 질량 중심 (COM)은 고정된 높이에서 움직인다. 즉 질량중심은 2차원상에서 움직인다고 전제한다. 따라서 보행시 상체의 모션을 제한하는것이 일반적이다.

2. 지면 반발력이 무게 중심을 기준으로 모멘트를 발생시키지 않는다. 이는 로봇이 균형을 유지하기위해 ZMP를 발바닥 영역 내에 유지할 수 있다는 것을 의미한다.

3. 선형 근사를 전제한다. 로봇의 움직임에 비선형성을 최대한 단순화하여 로봇의 움직임을 선형화한 모델로 해석한다.

2. 기구학적 해석 기본 개념 정리

- Rotation Matrix (회전 행렬) : 3차원상의 벡터를 원점 기반 축으로 돌리는 컨셉이다.

  일반적인 벡터해석과 마찬가지로 단위벡터로 분리해서 해석한다. 피연산 벡터와 회전축 벡터는 단위 벡터로 분해하고 회전축 벡터를 크기 1로 축척하고 해당 벡터를 단위벡터로 표현하면 단위벡터의 계수가 곧 해당 단위벡터가 속한 축과 회전축 사이각의 코사인 값이다. 유도는 내적을 활용했다.

  *예를 들어 회전축 벡터를 크기 1로 축척한 벡터가 a*i + b*j +c*k라고 할때 회전축과 X축이 이루는 각의 코사인 값은 a이다.*

  이런식으로 회전축 벡터를 기반으로각도를 유도하면 개별 각도를 확인할 수 있다.

벡터를 분해하고 개별 성분을 Z 축을 기준으로 프사이만큼 회전시킨다고 생각하면 다음과 같은 식이 유도됨을 쉽게 알 수 있다.

$$R_z(\psi) = \begin{bmatrix} cos(\psi) \\ sin(\psi) \\ 0 \end{bmatrix} \begin{bmatrix} -sin(\psi) \\ cos(\psi) \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

회전행렬은 다른 행렬에 곱하는 형태로 쓰이기 때문에 단위행렬(I)로부터 유도된 형태로 표현한다.

여기서 피연산 벡터를 단위벡터의 합인 단위행렬 I = i + j + k 라고 생각하고 개별 성분을

Z축을 기준으로 프사이 만큼 회전한다. 위 행렬에서 모든 열의 크기의 합이 1이 되는점에서 해당 내용을 재확인할 수 있다. 이때 최종 변환된 벡터는 1X3행렬로 피연산 벡터를 행렬로 표현하고 위 회전횡렬 공식을 곱하면 된다. 이를 벡터로 표현하려면 단순히 3x3행렬의 행방향으로 더한 값을 각 단위벡터의 계수로 채용하면 된다.

결론은 해당 행렬을 1X3 행렬로 표현된 피연산 벡터에 곱하면 1X3 행렬로 표현된 회전된 벡터 행렬을 얻을 수 있다.

이런 방법을 Y축 회전 , X 축 회전에도 역시 적용 할 수 있는데 곱해주는 회전행렬은 아래와 같다.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{bmatrix}$$

$$R_y(\phi) = \begin{bmatrix} cos(\phi) & 0 & sin(\phi) \\ 0 & 1 & 0 \\ -sin(\phi) & 0 & cos(\phi) \end{bmatrix}$$

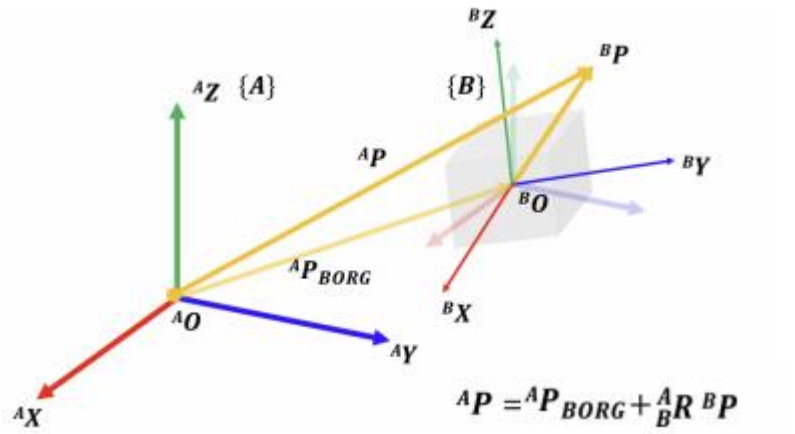
$$R_z(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Transformation Matrix(동차 변환 행렬) : 회전 행렬은 원점을 통과하는 회전축을 전제한다. 하지만 보편적인 회전축을 생각해보면 항상 회전축이 원점을 통과하지는 않는다. 이런 상황에서 일반적으로 원점에서 벡터 회전을 시키고 위치벡터를 더해 해당 원점을 밀어주는 컨셉을 사용할 수 있다. 이러한 계산을

직관적이고 쉽게 구현한 방법이 동차변환행렬이다. 동차 변환 행렬은 다음과 같은 구성을 갖는 4x4 행렬이다.

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \left[\begin{array}{c:c} \boldsymbol{R} & \boldsymbol{p}_0 \\ \hdashline 0 & 1 \end{array}\right] \begin{bmatrix} p_{x'} \\ p_{y'} \\ p_{z'} \\ 1 \end{bmatrix}$$

해당 행렬에서 R에는 3x3 회전 행렬이 들어가고 p0에는 3x1 위치 벡터 행렬이 들어간다. 쉽게 설명하면 원점에서 p0벡터 만큼 떨어진 점을 원점으로 하는 좌표계에서 정의된 피연산 벡터를 R 회전 행렬만큼 회전 시킨다고 생각하면 된다.



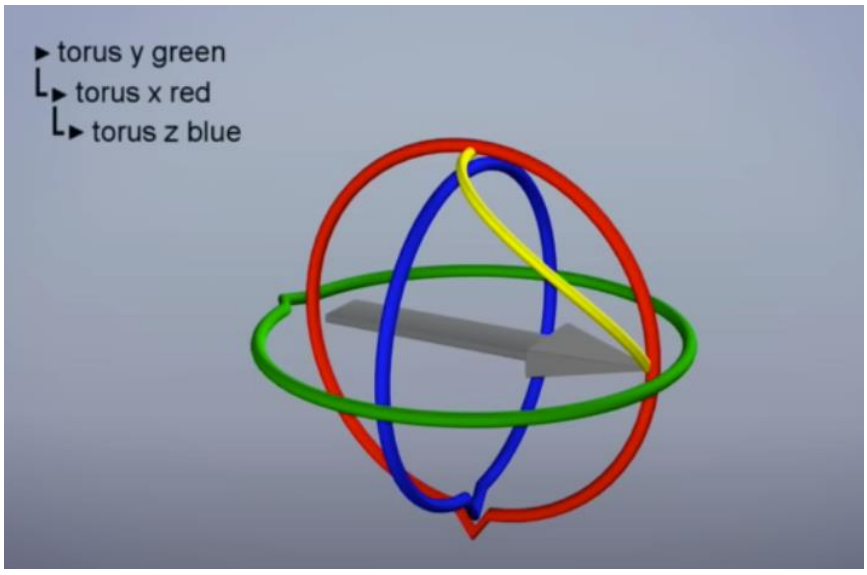
해당 서술을 기하적으로 표현하면 다음 그림처럼 표현할 수 있다. 이때 행렬곱과 행렬합의 연산 결과는 위 동차변환행렬을 곱한것과 같은 결과를 도출하므로 일반적으로 동차변환행렬을 곱하는 것으로 계산을 대체한다.

동차 변환행렬에서 4행에는 실제 벡터값이 아닌 수학적으로 합의된 숫자를 활용한다. 이는 해당 열이 위치벡터를 나타내면 1을 방향벡터를 나타내면 0을 적어 사용자가 쉽게 해당 식을 이해할 수 있게 돕는 역할을 한다. 뿐만 아니라 피연산행렬의 4행을 추가함으로 행렬곱이 성립하게하는 효과도 있다.

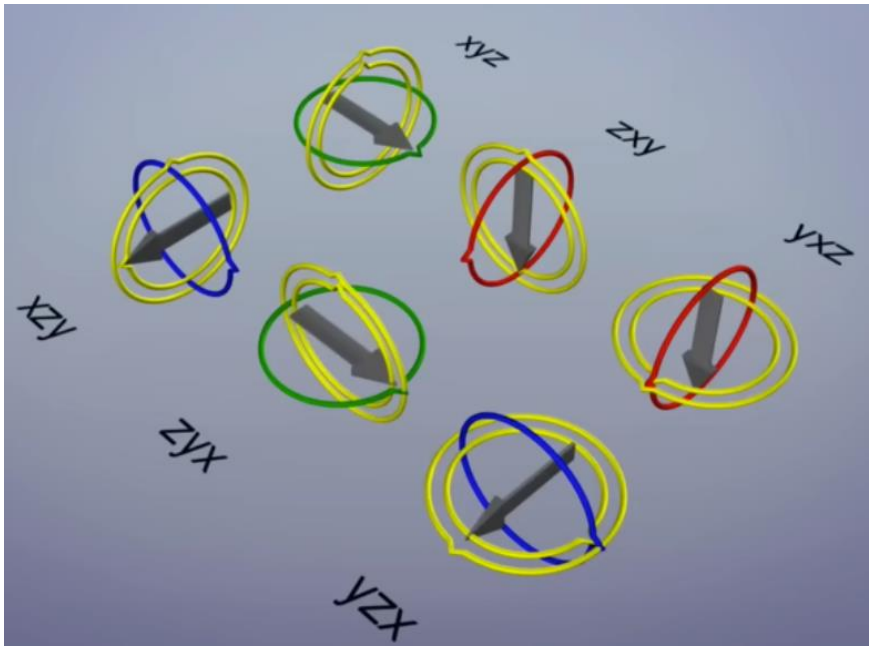
3. 짐벌락 현상

3-1 짐벌락 현상의 이해

짐벌락 현상은 짐벌이라는 기계적 구조에서 발생하는 문제이다. 축으로 회전운동을 표현할 때 3개의 링을 과하게 움직이면 짐벌 구조에서 2개의 회전축이 서로 평행하게 정렬되면서 자유도가 1개가 줄어드는 현상이다. 이런식으로 회전축이 겹친 상황에서 짐벌을 선형으로 움직이게 명령하면 다음과 같이 예상치 못한 곡선형 움직임이 나타날 수 있다.



이러한 현상은 축의 위계와 무관하게 발생한다.



3-2. 짐벌락 현상의 행렬식 관점의 이해

짐벌락 현상을 행렬식 관점에서 설명하겠다. 회전변환의 행렬이 특정 상태에서 랭크 손실을 일으킨다. 즉 행이나 열의 선형 독립성이 상실된다. 기본 개념을 정의하겠다.

- 행렬의 선형독립성: 행렬의 행이나 열이 서로 독립적으로 존재하면, 다른 행이나 열의 선형조합으로 표현될 수 없는 성질을 의미한다.

- 행렬의 랭크: 행렬 내에서 선형 독립적인 행 또는 열의 최대 개수를 의미한다.

짐벌락 현상이 발생하면 회전 행렬의 선형 독립성이 일부 축에서 사라진다. 즉 행렬의 랭크가 손실된다. 이는 오일러 각의 한계를 들어내는 현상이다.

4. 동차 변환 행렬 예제 풀이

- 문제 조건 :

• [x,y,th]로 위치정보를 표현할 때(th는 xy평면에서 바라보는 방향) A, B, C가 각각 A[3,4,45°], B[-6,7,-60°], C[10,2,135°]의 위치를 가질 때 A->B로의 동차변환행렬과 C->B로의 동차변환 행렬만을 가지고 A->C로의 이동을 Eigen으로 표현.

- 수식적 풀이 과정:

1. 위치 정보 행렬을 동차변환행렬로 변환한다.

(1) A의 동차변환행렬 (T_A)

A의 위치 정보: $[3, 4, 45^\circ]$

$$T_A = \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 3 \\ \sin 45^\circ & \cos 45^\circ & 4 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 3 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 4 \\ 0 & 0 & 1 \end{bmatrix}$$

(2) B의 동차변환행렬 (T_B)

B의 위치 정보: $[-6, 7, -60^\circ]$

$$T_B = \begin{bmatrix} \cos(-60^\circ) & -\sin(-60^\circ) & -6 \\ \sin(-60^\circ) & \cos(-60^\circ) & 7 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{\sqrt{3}}{2} & -6 \\ -\frac{\sqrt{3}}{2} & \frac{1}{2} & 7 \\ 0 & 0 & 1 \end{bmatrix}$$

(3) C의 동차변환행렬 (T_C)

C의 위치 정보: $[10, 2, 135^\circ]$

$$T_C = \begin{bmatrix} \cos 135^\circ & -\sin 135^\circ & 10 \\ \sin 135^\circ & \cos 135^\circ & 2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 10 \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

2. 해당 값을 바탕으로 A->B로의 동차변환행렬과 C->B로의 동차변환행렬로 변환한다.

1. **A->B로의 동차변환행렬 (T_{AB}):** $T_{AB} = T_B \cdot T_A^{-1}$
2. **C->B로의 동차변환행렬 (T_{CB}):** $T_{CB} = T_B \cdot T_C^{-1}$

3. Tcb 행렬에 Tab 행렬의 역행렬 변환을 한 행렬을 곱하면 원하는 Tac 행렬을 찾을 수 있다.

- 시행 결과

```
lwj@lwj:~/TransformMatrixEigen/build$ ./transform

A Transformation Matrix (T_A):
 0.707107 -0.707107         3
 0.707107  0.707107         4
        0         0         1
B Transformation Matrix (T_B):
      0.5  0.866025        -6
-0.866025       0.5         7
        0         0         1
C Transformation Matrix (T_C):
-0.707107 -0.707107        10
 0.707107 -0.707107         2
        0         0         1
A->B Transformation Matrix (T_A_B):
-0.258819  0.965926  -9.08725
-0.965926 -0.258819   10.9331
        0         0         1
C->B Transformation Matrix (T_C_B):
-0.965926 -0.258819    4.1769
 0.258819 -0.965926   6.34366
        0         0         1
A->C Transformation Matrix (T_A_C):
 0 -1 14
 1  0 -1
 0  0  1
```

- 소스코드

```cpp
#include <iostream>
#include <Eigen/Dense>
#include <cmath>

#define DEG2RAD(angle) ((angle) * M_PI / 180.0)

using namespace std;
using namespace Eigen;

// 동차변환 행렬 생성 함수
Matrix3d createTransformMatrix(double x, double y, double theta_deg)
{
double theta = DEG2RAD(theta_deg); // 각도를 라디안으로 변환
Matrix3d T = Matrix3d::Identity(); // 3x3 단위 행렬 생성
T(0, 0) = cos(theta);
T(0, 1) = -sin(theta);
T(1, 0) = sin(theta);
T(1, 1) = cos(theta);
T(0, 2) = x; // 평행 이동 x
T(1, 2) = y; // 평행 이동 y
return T;
}

int main()
{
// A, B, C의 위치와 방향 [x, y, theta]
Vector3d A(3, 4, 45); // A: (3, 4) 위치, 45도 방향
Vector3d B(-6, 7, -60); // B: (-6, 7) 위치, -60도 방향
Vector3d C(10, 2, 135); // C: (10, 2) 위치, 135도 방향

// A, B, C의 동차변환 행렬 생성
Matrix3d T_A = createTransformMatrix(A(0), A(1), A(2));
Matrix3d T_B = createTransformMatrix(B(0), B(1), B(2));
Matrix3d T_C = createTransformMatrix(C(0), C(1), C(2));

// A->B 변환 행렬 계산
Matrix3d T_A_B = T_B * T_A.inverse();

// C->B 변환 행렬 계산
Matrix3d T_C_B = T_B * T_C.inverse();

// A->C 변환 행렬 계산
```

```
Matrix3d T_A_C = T_C_B.inverse() * T_A_B;


// 결과 출력
cout << "A Transformation Matrix (T_A):\n" << T_A << endl;
cout << "B Transformation Matrix (T_B):\n" << T_B << endl;
cout << "C Transformation Matrix (T_C):\n" << T_C << endl;
cout << "A->B Transformation Matrix (T_A_B):\n" << T_A_B << endl;
cout << "C->B Transformation Matrix (T_C_B):\n" << T_C_B << endl;
cout << "A->C Transformation Matrix (T_A_C):\n" << T_A_C << endl;


return 0;
}
```