


보행1일차

HW1

HW1


1) x 축 기준으로 회전



$$\begin{aligned} z' &= y \sin \theta + z \cos \theta \\ y' &= y \cos \theta - z \sin \theta \\ \begin{bmatrix} y' \\ z' \end{bmatrix} &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} \end{aligned}$$

$$\begin{aligned} R_x &= R_x \begin{bmatrix} x \\ 0 \\ 0 \end{bmatrix} + R_x \begin{bmatrix} 0 \\ y \\ 0 \end{bmatrix} + R_x \begin{bmatrix} 0 \\ 0 \\ z \end{bmatrix} \\ &= R_x \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} x + R_x \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} y + R_x \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} z \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \end{aligned}$$

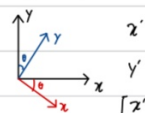
2) y 축 기준 회전



$$\begin{aligned} x' &= x \cos \theta + z \sin \theta \\ z' &= -x \sin \theta + z \cos \theta \\ \begin{bmatrix} x' \\ z' \end{bmatrix} &= \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} \end{aligned}$$

$$\begin{aligned} R_y &= R_y \begin{bmatrix} x \\ 0 \\ 0 \end{bmatrix} + R_y \begin{bmatrix} 0 \\ y \\ 0 \end{bmatrix} + R_y \begin{bmatrix} 0 \\ 0 \\ z \end{bmatrix} \\ &= R_y \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} x + R_y \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} y + R_y \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} z \\ &= \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \end{aligned}$$

3) z 축 기준으로 회전



$$\begin{aligned} x' &= x \cos \theta - y \sin \theta \\ y' &= x \sin \theta + y \cos \theta \\ \begin{bmatrix} x' \\ y' \end{bmatrix} &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \end{aligned}$$

$$\begin{aligned} R_z &= R_z \begin{bmatrix} x \\ 0 \\ 0 \end{bmatrix} + R_z \begin{bmatrix} 0 \\ y \\ 0 \end{bmatrix} + R_z \begin{bmatrix} 0 \\ 0 \\ z \end{bmatrix} \\ &= R_z \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} x + R_z \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} y + R_z \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} z \\ &= \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \end{aligned}$$

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

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

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

짐벌락 현상

⇒ 로봇의 관절이 특정한 각도에 도달했을 때, 회전 자유도가 상실되어 의도한 방향으로 움직이지 못하게 되는 문제이다.

- 짐벌락 현상이 발생하는 이유

짐벌락은 주로 오일러 각을 이용한 3D 회전 표현에서 발생한다. 오일러 각은 세 가지 축을 기준으로 연속적인 회전을 표현하는 방식인데, 특정 각도에서 두 축이 정렬되어 하나의 축처럼 움직이는 문제가 발생할 수 있다. 이럴 경우 원래는 세 축 각각의 회전을 조절할 수 있어야 하지만, 특정 각도에서 축 하나의 회전 정보가 상실되어 두 축만 남아 원하는 방향 제어가 어렵게 된다.

- 짐벌락이 휴머노이드 보행에 미치는 영향

휴머노이드 보행은 **정확한 균형과 방향 제어**가 중요하다. 예를 들어 다리와 허리의 각도에 따라 로봇이 앞으로 걷거나 좌우로 방향을 바꿀 수 있다. 그러나 짐벌락이 발생하면 한 축의 회전이 무의미해지거나 제한되면서 보행 도중 방향 제어가 어렵고 불안정한 동작을 일으킬 수 있다. 특히 바닥에서 미끄러지거나 불안정한 지면에서 더 쉽게 넘어질 위험이 있다.

- 행렬식(Determinant)

행렬식은 **행렬의 특성을 나타내는 값**으로, 특히 회전 및 변환 행렬에서 중요한 의미를 가진다. 3x3 회전 행렬의 행렬식이 0이 아니면 그 행렬은 **정규 변환**을 나타내며 짐벌락을 피할 수 있다. 반면에 행렬식이 0에 가까워지면 짐벌락 상태가 되어 로봇이 특정 방향으로 움직이지 못하게 된다.

- 역행렬(Inverse Matrix)

역행렬은 주어진 행렬을 역변환하는 역할을 한다. 예를 들어, 로봇의 현재 자세를 나타내는 회전 행렬의 역행렬을 사용하면 원래의 자세로 되돌릴 수 있다. 짐벌락 상태에서는 회전 행렬이 역행렬을 가지지 못할 수 있으며, 이는 로봇이 **자유롭게 회전하지 못하게** 만든다.

- 행렬의 랭크(Rank)

행렬의 랭크는 **행렬의 선형 독립성**을 나타내며, 로봇이 다양한 방향으로 회전할 수 있는지를 결정한다. 3x3 회전 행렬의 랭크가 3이면 세 축 모두 독립적으로 회전할 수 있음을 의미하지만, 짐벌락이 발생하면 랭크가 줄어들어 자유도가 감소한다. 즉, 짐벌락 상태에서는 한 축이 다른 축에 의존하게 되어 두 축만으로 회전을 표현할 수밖에 없게 된다.

COM(Center of Mass, 질량 중심)

COM은 로봇의 전체 질량이 집중된 가상의 위치로, **로봇의 무게 중심**이라 할 수 있다. 로봇이 안정적으로 서 있거나 걸을 때, COM의 위치는 로봇의 균형을 결정짓는 중요한 요소이다.

- **의미**: COM은 로봇의 모든 질량 요소가 집중된 지점으로 볼 수 있으며, 로봇의 각 관절과 부위의 위치와 질량을 고려해 계산된다.
- **안정성**: 일반적으로 로봇이 안정된 자세를 유지하기 위해서는 COM가 지면과 닿는 지지면(base of support) 내부에 있어야 한다.
- **제어**: 보행 제어 알고리즘은 COM의 위치를 추적하여, 로봇이 앞으로 넘어지거나 옆으로 기울어지지 않도록 균형을 맞춘다.

COP (Center of Pressure, 압력 중심)

COP는 로봇이 지면에 가하는 **압력의 중심점**으로, 실제 발이 닿는 지점에서의 압력을 기반으로 계산된다. 즉, 로봇이 서 있는 지면에서 가해지는 힘의 평형점을 나타낸다.

- **의미:** COP는 로봇이 지면과 접촉하는 부분에서 압력의 균형을 나타내며, 보통 발바닥에 있는 힘 센서로 측정한다.
- **안정성:** COP는 지면과의 접촉 면적 안에 있어야 안정적인 자세가 유지된다. 보행 중에 COP가 움직임에 맞춰 변동되며, 안정성을 위해 보통 ZMP와 일치하도록 조정한다.
- **제어:** 로봇의 균형을 유지하기 위해 COP의 위치를 지속적으로 조정하며, 이는 보행 중의 발 위치 및 각도 조정으로 이루어진다.

ZMP (Zero Moment Point, 영 모멘트 지점)

ZMP는 로봇이 보행하는 동안 **넘어지지 않도록 지면에 가하는 힘의 평형점**으로, 로봇의 균형 상태를 판단하는 중요한 지표이다. ZMP가 로봇의 지지면 내부에 존재하면 안정적인 보행이 가능하다고 간주한다.

- **의미:** ZMP는 로봇의 무게와 가속도에 의해 발생하는 모멘트가 0이 되는 지점으로, 로봇이 균형을 잡는 데 필요한 기준점이다. 로봇의 중심이 이 지점을 지나면 모멘트가 발생하지 않아 로봇이 넘어지지 않는다.
- **안정성:** ZMP가 지면에 닿는 지지면의 범위 내에 존재해야 로봇이 넘어지지 않고 균형을 유지할 수 있다. 보행 중 ZMP가 지지면 밖으로 벗어나면 균형을 잃어 넘어지게 된다.
- **제어:** ZMP 제어는 보행 제어의 핵심 요소로, 주로 COM의 움직임을 조절하여 ZMP가 항상 지지면 내에 있도록 한다.

COM, COP, ZMP의 관계

- **보행 중 안정성 유지:** 로봇이 보행할 때 ZMP와 COP는 유사한 궤적을 따르며, COM은 ZMP와의 관계를 통해 안정성을 판단한다.
- **균형 유지 전략:** 로봇 제어 알고리즘은 ZMP와 COP가 지지면 내부에 있도록 조절하며, 보행 중에 COM 위치가 이들 지점과 조화를 이루도록 만든다.

Capture Point(CP)기반 실시간 패턴생성(LIPM)

개념

- **Capture Point (CP):**
CP는 로봇이 쓰러지지 않고 균형을 회복하기 위해 발을 딛어야 할 위치를 의미한다. 즉, 만약 로봇이 쓰러질 것 같다면, 로봇의 발을 CP로 옮겨놓으면 다시 균형을 유지할 수 있다.
- **LIPM (Linear Inverted Pendulum Model):**
LIPM은 로봇의 보행 모델링을 위해 CoM을 단순한 역진자 형태로 가정하여, 로봇의 움직임을 계산하는 기법이다. LIPM에서는 CoM이 일정한 높이에서 움직이며, 주로 중력을 고려해 간단한 2차원 운동 방정식으로 표현된다.

작동 방식

Capture Point 기반 LIPM 보행에서, 로봇은 현재 CoM의 위치와 속도를 통해 CP를 예측하고, 보행을 통해 CoM이 계속해서 균형을 유지하도록 발걸음 위치를 조정한다. 만약 균형을 유지하기 힘든 상황이라면, CP가 바깥쪽으로 계산되면서 로봇이 자연스럽게 발을 내디뎌 넘어지지 않도록 한다.

장점

- 적응성: 실시간으로 CP를 계산해 외부 요인에 적응하므로 예측하지 못한 충격에도 대응 가능.
- 안정적인 보행: CoM과 CP의 관계를 실시간으로 조정해 균형을 유지하며 자연스러운 보행 패턴을 생성.
- 실시간 반응: 실시간으로 보행 패턴을 생성해 전통적인 고정된 보행 궤적을 따르는 방식보다 안전하고 유연하게 대처할 수 있음.

비선형Preview control(MPC)

개요

MPC는 제어 시스템이

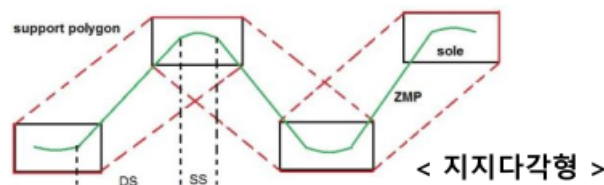
예측 모델을 이용해 일정 시간 후의 로봇 상태를 미리 계산하고, 목표에 도달하는 최적의 경로를 실시간으로 찾아내는 방식이다. 비선형 시스템에서 사용하는 MPC는 **비선형 Preview Control**이라고 불리며, 특히 로봇 보행에서 안정성을 높이는 데 사용된다.

작동원리

1. 미래 상태 예측: 현재 상태와 제어 입력에 따라 일정 시간 후의 로봇 위치와 속도를 예측한다. 여기에는 주로 보행 궤적의 예상 위치, 속도, 방향이 포함된다.
2. 비선형 모델 사용: 보행 로봇의 움직임은 중력, 마찰, 관성 등 비선형적 요소의 영향을 받으므로 비선형 시스템 모델을 사용해 보다 정확한 예측이 가능하다. 예를 들어, LIPM 모델이나 개선된 비선형 모델을 적용해 로봇이 걷는 중에 발생하는 동적 변화를 반영한다.
3. 미리보기 제어: MPC는 미래의 여러 시점을 미리 고려하는 방식으로, 예측되는 상태와 목표 상태 사이의 오차를 줄이기 위해 최적의 제어 입력을 계산한다. 예를 들어, 로봇이 목표로 삼고 있는 발 디딤 위치나 균형 유지에 필요한 보정 위치를 계산한다.
4. 실시간 최적화: 예측된 정보를 바탕으로 로봇이 실시간으로 균형을 유지할 수 있도록 최적의 제어 입력을 계산하여 적용한다. 이 과정은 반복되며, 매 순간 새로운 예측과 최적화가 이루어진다.

보행 알고리즘

로봇 보행 알고리즘에서 보행 궤적 생성 과정은 안정적으로 목표 지점까지 이동할 수 있도록 발 위치와 균형을 설계하는 것이다. 이 과정은 로봇의 발 위치(foot_step좌표)를 목표로 하여 지지다각형과 선형 역진자 모델 기반으로 ZMP와 COM 궤적을 설계하고 로봇에 적용해 보행 궤적을 완성한다.



로봇의 목표 foot_step좌표 입력 ⇒ LIPM 기반 COM궤적 설계 ⇒ 생성된 foot_step좌표를 로봇에 적용하여 보행

• DSP (Double Support Phase: 양발 지지 구간)

- 정의: DSP는 보행 과정 중 두 발이 동시에 지면에 닿아 있는 시간 구간이다. 즉, 양쪽 발이 동시에 지면에 닿아 있어, 로봇이 두 다리로 몸을 지탱하게 된다.

- **역할:** DSP는 **로봇이 안정성을 최대한으로 확보하는 구간**으로, 보행 중 넘어지지 않도록 균형을 잡는 데 중요한 역할을 하난. DSP가 길수록 로봇은 안정적인 자세를 유지하기 쉬워진다.
- **장점:** 두 발이 지면에 닿아 있어 무게 중심의 변동에 대한 안정성이 높아지며, 외부에서 오는 힘이나 외란에 대한 저항력이 커진다.

• SSP (Single Support Phase: 단일 발 지지 구간)

- **정의:** SSP는 **보행 과정 중 한쪽 다리만 지면을 지지하고 있는 시간 구간**이다. 반대쪽 다리는 다음 발걸음을 위해 공중에 들어 올려 이동하게 된다.
- **역할:** SSP는 로봇이 한 발로 몸을 지탱하고, 다른 발은 목표 지점으로 이동하는 구간이다. 보행의 추진력과 속도는 주로 SSP 동안 결정되며, 로봇의 이동을 위한 필수 단계이다.
- **장점:** SSP는 다음 발을 움직일 수 있도록 하여 보행을 지속하게 해 준다. 다만, SSP 동안에는 로봇이 한 발로만 균형을 잡아야 하므로 안정성이 낮아질 수 있다.

Frames (좌표계)

- 각 물체는 독립적인 좌표계를 가질 수 있으며, 이를 통해 물체의 위치와 방향을 특정 좌표계에서 표현할 수 있다.
- 예시로 로봇 팔과 같은 시스템에서 각 관절이나 부품의 위치를 다른 기준 좌표계로 변환할 때 이러한 좌표계 개념이 사용된다.

Rotation Matrix (회전 행렬)

- 특정 축을 기준으로 회전할 때의 회전 행렬을 정의하며, 회전각을 통해 물체의 방향을 설정할 수 있다.
- 예를 들어, z 축을 기준으로 45도 회전하는 경우, 이를 수식으로 나타내는 방법이 설명된다.
- 각 축 (x, y, z)에 대해 독립적인 회전 행렬 (R_x , R_y , R_z)을 정의하고, 이를 조합하여 복잡한 회전을 표현할 수 있다.

Rotation Matrix - 축 회전

- 다음 슬라이드들에서는 개별 축에 대해 회전하는 경우를 수식으로 나타냅니다.
- $R_x(\theta)$: x축 기준 회전 행렬
- $R_y(\phi)$: y축 기준 회전 행렬
- $R_z(\psi)$: z축 기준 회전 행렬
- 이 회전 행렬을 사용하여 좌표를 변환할 수 있으며, 여러 회전을 조합해 최종적인 회전 행렬 $R=R_zR_yR_x$ 을 계산합니다.

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

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

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

Transformation (변환)

- 변환은 단순한 회전뿐만 아니라 이동(translation)까지 포함하는 개념이다.
- 3차원 공간에서의 물체의 위치를 다른 좌표계로 변환하기 위해서는 변환 행렬 (Transformation Matrix)이 필요하다.
- 변환 행렬은 회전 행렬과 이동 벡터를 합쳐 표현할 수 있으며, 이를 통해 물체의 위치와 방향을 동시에 변환할 수 있다.

동차변환행렬(Homogeneous Transformation Matrix)과 **DH 파라미터**(Denavit-Hartenberg Parameters)는 로봇 매니퓰레이터의 각 관절과 링크 간의 관계를 표현하는 데 중요한 도구이다. DH 파라미터는 매니퓰레이터의 각 링크 간의 상대적인 위치와 회전 정보를 체계적으로 표현하며, 이를 동차변환행렬로 나타내어 로봇의 모든 관절 위치와 자세를 쉽게 계산할 수 있다.

동차변환행렬

⇒ 3차원 공간에서 **물체의 위치와 자세**를 동시에 표현하는 데 사용되는 4x4행렬이다. 이 행렬은 물체가 한 좌표계에서 다른 좌표계로 변환될 때 **회전과 이동**을 하나의 행렬로 통합하여 계산할 수 있게 해준다.

DH 파라미터(DH Parameters)

DH 파라미터는 각 **링크의 위치와 방향을 정의하는 네 가지 변수**로 구성된다. 이 변수들은 각 링크와 다음 링크 사이의 상대적인 변환을 표현한다. 일반적으로 DH 파라미터는 다음 네 가지 값으로 정의된다.

- θ_i : 링크 i 에서의 회전 각도 (z 축을 중심으로 회전)
- d_i : 링크 i 의 길이 (z 축을 따라 이동하는 거리)
- a_i : 링크 i 와 링크 $i + 1$ 사이의 거리 (x 축을 따라 이동)
- α_i : 링크 i 의 꼬임 각도 (x 축을 중심으로 회전)

이러한 DH 파라미터는 **DH 변환행렬**을 이용하여 각 링크 간의 변환을 행렬로 나타내는 데 사용된다.

DH 파라미터를 통한 동차변환행렬

각 링크 i 에서 링크 $i+1$ 로의 위치와 자세 변환은 다음과 같은 **동차변환행렬**로 표현할 수 있다.

$$T_i^{i+1} = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \cos \alpha_i & \sin \theta_i \sin \alpha_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \theta_i \cos \alpha_i & -\cos \theta_i \sin \alpha_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

이 행렬은 다음과 같이 각 요소를 나타낸다:

- **회전**: θ_i 와 α_i 는 링크 간의 회전 관계를 표현.
- **이동**: a_i 와 d_i 는 링크 간의 이동 관계를 나타냄.

DH 파라미터 분석 절차

1. **좌표축 정의**: 각 링크의 끝 지점에 좌표계를 설정하며, z_i 축을 다음 링크로의 회전 축으로 설정.
2. **파라미터 값 지정**: 각 링크에 대해 θ_i , d_i , a_i , α_i 값을 지정.
3. **동차변환행렬 계산**: 각 링크의 DH 파라미터를 통해 T_i^{i+1} 행렬을 계산하여 각 링크 사이의 변환을 구함.
4. **총 변환 행렬 계산**: 각 변환 행렬을 곱하여 로봇의 베이스 프레임에서 각 관절과 말단 위치까지의 변환을 계산.

DH 파라미터를 통한 로봇 자세 계산

로봇의 모든 링크에 대해 변환 행렬을 곱하여 최종적으로 **기준 프레임에서 말단 프레임까지의 위치와 자세**를 구할 수 있다.

링크가 3개인 경우

$$T_0^3 = T_0^1 \cdot T_1^2 \cdot T_2^3$$

이를 통해 로봇의 특정 위치와 자세를 제어할 수 있다.

DH 파라미터의 장점과 한계

- **장점:** DH 파라미터는 링크 간의 변환을 표준화하여 복잡한 계산을 간소화하며, 로봇 매니퓰레이터의 모델링에 매우 유용하다.
- **한계:** 모든 로봇 구조에 유연하게 적용되지 않을 수 있으며, 특히 비정형 구조나 비정형 회전축을 가진 로봇의 경우 추가적인 변환이 필요하다.

동차변환행렬과 DH 파라미터를 결합하여, 로봇의 각 관절 위치와 방향을 직관적이고 효율적으로 계산할 수 있다. 이는 로봇 제어와 경로 계획에서 매우 중요한 역할을 한다

HW2

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

#define PI 3.141592653589793238463

Eigen::Matrix4d createHomogeneousMatrix(double x, double y, double theta) {
    Eigen::Matrix4d matrix = Eigen::Matrix4d::Identity(); // 단위 행렬 생성

    double rad = theta * PI / 180.0; // 각도를 라디안으로 변환

    // 회전 부분 설정
    matrix(0,0) = cos(rad);
    matrix(0,1) = -sin(rad);
    matrix(1,0) = sin(rad);
    matrix(1,1) = cos(rad);

    // 평행이동 부분 설정
    matrix(0,3) = x;
    matrix(1,3) = y;

    return matrix; // 변환 행렬 반환
}

void printTransform(const Eigen::Matrix4d& T, const std::string& name) {
    std::cout << name << ":\n";
    std::cout << "Matrix:\n" << T << "\n";
    std::cout << "Position (x,y): " << T(0,3) << ", " << T(1,3) << "\n";
    double theta = atan2(T(1,0), T(0,0)) * 180.0 / PI;
    std::cout << "Rotation (degrees): " << theta << "\n\n";
}

int main() {
    // 포즈 생성
```



```

Eigen::Matrix4d T_A = createHomogeneousMatrix(3, 4, 45);
Eigen::Matrix4d T_B = createHomogeneousMatrix(-6, 7, -60);
Eigen::Matrix4d T_C = createHomogeneousMatrix(10, 2, 135);

Eigen::Matrix4d T_AB = T_B.inverse() * T_A; // A -> B
Eigen::Matrix4d T_CB = T_B.inverse() * T_C; // C -> B

std::cout << "Given transforms:\n";
printTransform(T_AB, "T_AB (A to B transform)");
printTransform(T_CB, "T_CB (C to B transform)");

Eigen::Matrix4d T_AC = T_CB.inverse() * T_AB;

std::cout << "\nCalculated transform:\n";
printTransform(T_AC, "T_AC (A to C transform)");

Eigen::Vector4d p_A(1, 0, 0, 1); // A 좌표계의 테스트 점 (1, 0)
Eigen::Vector4d origin_A(0, 0, 0, 1); // A 좌표계의 원점

Eigen::Vector4d p_C = T_AC * p_A;
Eigen::Vector4d origin_C = T_AC * origin_A;

std::cout << "Points in A's frame:\n";
std::cout << "Origin: " << origin_A.transpose() << "\n";
std::cout << "Test point: " << p_A.transpose() << "\n\n";

std::cout << "Points in C's frame:\n";
std::cout << "A's origin: " << origin_C.transpose() << "\n";
std::cout << "Test point: " << p_C.transpose() << "\n";

return 0;
}

```

결과

```

Given transforms:
T_AB (A to B transform):
Matrix:
-0.258819 -0.965926      0    7.09808
0.965926 -0.258819      0    6.29423
0          0          1      0
0          0          0      1
Position (x,y): 7.09808, 6.29423
Rotation (degrees): 105

T_CB (C to B transform):
Matrix:
-0.965926  0.258819      0    12.3301

```

```
-0.258819 -0.965926      0  11.3564
0          0          1      0
0          0          0      1
Position (x,y): 12.3301, 11.3564
Rotation (degrees): -165
```

Calculated transform:
T_AC (A to C transform):
Matrix:

```
0      1      0  6.36396
-1      0      0  3.53553
0      0      1      0
0      0      0      1
Position (x,y): 6.36396, 3.53553
Rotation (degrees): -90
```

Points in A's frame:
Origin: 0 0 0 1
Test point: 1 0 0 1

Points in C's frame:
A's origin: 6.36396 3.53553 0 1
Test point: 6.36396 2.53553 0 1

결과를 검증:

1. 회전 검증:

- A는 $45^\circ \rightarrow$ B는 -60° (T_{AB} 의 각도: $105^\circ = -60^\circ - 45^\circ$)
- C는 $135^\circ \rightarrow$ B는 -60° (T_{CB} 의 각도: $-165^\circ = -60^\circ - 135^\circ$)
- 결과적으로 $A \rightarrow C$ 의 회전은 -90° ($135^\circ - 45^\circ = 90^\circ$)가 나와야 하고, 실제로 그렇게 나왔다.

2. 포지션 검증:

- A의 원점이 C 프레임에서 (6.36396, 3.53553)
- (1,0,0) 점이 C 프레임에서 (6.36396, 2.53553)

3. 변환 행렬의 특성:

- 회전 행렬 부분이 정규직교($\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$)로 올바르게 나왔다
- $T_{AC} = T_{CB}^{(-1)} * T_{AB}$ 의 계산이 올바르게 수행되었다

4. 테스트 포인트 검증:

- A프레임의 (1,0,0)점이 -90° 회전했을 때 (0,-1,0)이 되어야 하고
- 이것이 A의 원점 위치에 더해져서 최종 위치가 된다
- y좌표가 $3.53553 - 1 = 2.53553$ 으로 계산되는 것이 맞다

```

Eigen::Matrix4d createHomogeneousMatrix(double x, double y, double theta) {
    Eigen::Matrix4d matrix = Eigen::Matrix4d::Identity(); // 단위 행렬 생성
    double rad = theta * PI / 180.0; // 각도를 라디안으로 변환

    // 회전 부분 설정
    matrix(0,0) = cos(rad);
    matrix(0,1) = -sin(rad);
    matrix(1,0) = sin(rad);
    matrix(1,1) = cos(rad);

    // 평행이동 부분 설정
    matrix(0,3) = x;
    matrix(1,3) = y;

    return matrix; // 변환 행렬 반환
}

```

- `matrix` 는 처음에 단위 행렬로 설정된다.
- 각도 `theta` 는 **라디안**으로 변환되어, `cos` 와 `sin` 을 이용해 회전 부분을 설정한다.
- `(x, y)` 평행이동을 마지막 열에 추가한다.
- 이 함수는 최종적으로 회전과 평행이동을 포함하는 변환 행렬을 반환한다.