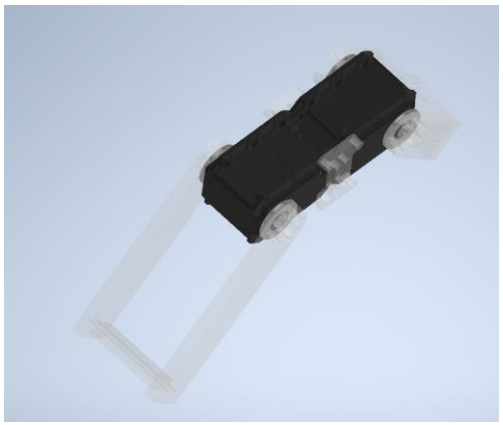
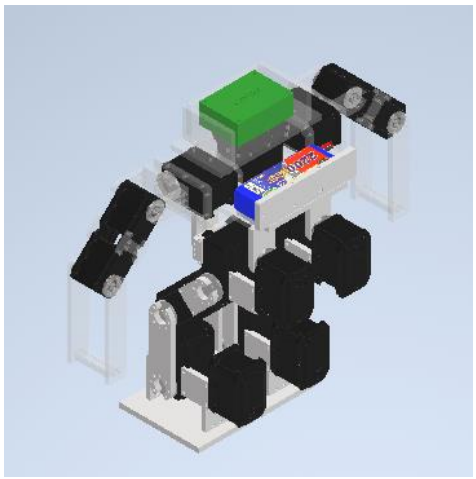
격투로봇팔 기구학 해석

19기 예비단원 이원준

1. 격투 로봇 팔 구성





- 실제 매니퓰레이터는 몸체쪽 부터 피치축 -> 링크 1번 (길이 36 mm) -> 롤축 1번 -> 링크 2번 (길이 77.3mm) -> 롤축 2번 -> 링크 3번 (100mm)로 구성 되어 있다.

- 매트랩에서는 첫번째 로테이션 조인트가 바닥을 향하도록 매니퓰레이터를 시계 방향으로 90도 회전 시켜서 요 축 -> 링크 1번 (길이 36 mm) -> 롤 축 1번 -> 링크 2번 (길이 77.3mm) -> 롤 축 2번 -> 링크 3번 (100mm)으로 구성하겠다.

2. 순기구학 해석 코드

```
clc;
clear;
close all;

%% 기본 설정
% 링크 길이 (mm 단위)
link_lengths = [36, 77.2, 100]; % 링크 1, 2, 3의 길이

% 초기 조인트 각도 (라디안 단위)
theta1 = deg2rad(0); % 조인트 1 초기화
theta2 = deg2rad(0); % 조인트 2 초기화
theta3 = deg2rad(0); % 조인트 3 초기화

% 조인트 개수
n = 3; % 총 3개의 조인트

% 순기구학 계산 함수
T_single = @(theta, d, a, alpha) [
    cos(theta), -sin(theta)*cos(alpha),  sin(theta)*sin(alpha), a*cos(theta);
    sin(theta),  cos(theta)*cos(alpha), -cos(theta)*sin(alpha), a*sin(theta);
    0,           sin(alpha),            cos(alpha),             d;
    0,           0,                     0,                      1;
];

%% 매니퓰레이터 초기 설정
figure; % 그래프를 출력할 새 창 생성
updateManipulator(theta1, theta2, theta3);

disp('매니퓰레이터 초기 설정값 설정완료했습니다.');
disp('조인트 각도를 입력하여 매니퓰레이터를 업데이트하세요.');
disp('예: theta1 = 30; theta2 = 45; theta3 = 60;');

%% 사용자 입력 대기 및 업데이트 루프
while true
    % 사용자 입력 받기
    user_input = input('각도를 입력하세요 (exit 입력 시 종료): ', 's');

    % 종료 조건
    if strcmpi(user_input, 'exit')
        disp('프로그램을 종료합니다.');
        break;
    end

    % 사용자 입력 실행
    try
        % 사용자 입력 실행 및 업데이트
        eval(user_input); % 입력된 명령 실행
        theta1 = deg2rad(theta1); % 각도를 라디안으로 변환
        theta2 = deg2rad(theta2);
        theta3 = deg2rad(theta3);

        % 매니퓰레이터 업데이트
        updateManipulator(theta1, theta2, theta3);
    catch
        disp('잘못된 입력입니다. 다시 입력하세요.');
    end
end

%% 매니퓰레이터 업데이트 함수
function updateManipulator(theta1, theta2, theta3)
    % 링크 길이
    link_lengths = [36, 77.2, 100];

    % DH 테이블 업데이트
    DH_table = [
        theta1, link_lengths(1), 0,      pi/2;
        theta2, 0,               link_lengths(2), 0;
        theta3, 0,               link_lengths(3), 0;
    ];

    % 순기구학 계산
    T = eye(4); % 초기 변환 행렬
    positions = [0, 0, 0]; % 기본 좌표계의 원점

    for i = 1:size(DH_table, 1)
        theta = DH_table(i, 1);
        d = DH_table(i, 2);
        a = DH_table(i, 3);
        alpha = DH_table(i, 4);

        % 현재 조인트의 변환 행렬 계산
        T_i = T_single(theta, d, a, alpha);
        T = T * T_i; % 누적 변환 행렬
        positions = [positions; T(1:3, 4)']; % 각 조인트 위치 저장
    end

    % 매니퓰레이터 그래프 업데이트
    plot3(positions(:, 1), positions(:, 2), positions(:, 3), '-o', 'LineWidth', 2);
    grid on;
    xlabel('X (mm)');
    ylabel('Y (mm)');
    zlabel('Z (mm)');
    title('Manipulator Forward Kinematics');
    axis equal;
    view(3); % 3D 뷰 설정
    drawnow; % 그래프 업데이트
end
```

```
clc;
clear;
close all;
%% 기본 설정
% 링크 길이 (mm 단위)
link_lengths = [36, 77.2, 100]; % 링크 1, 2, 3의 길이
% 초기 조인트 각도 (라디안 단위)
theta1 = deg2rad(0); % 조인트 1 초기값
theta2 = deg2rad(0); % 조인트 2 초기값
theta3 = deg2rad(0); % 조인트 3 초기값
% 조인트 개수
n = 3; % 총 3개의 조인트
%% 매니퓰레이터 초기 생성
figure; % 그래프를 출력할 새 창 생성
updateManipulator(theta1, theta2, theta3);
disp('매니퓰레이터 초기 상태가 생성되었습니다.');
disp('조인트 각도를 입력하여 매니퓰레이터를 업데이트하세요.');
disp('예: theta1 = 30; theta2 = 45; theta3 = 60;');
%% 사용자 입력 대기 및 업데이트 루프
while true
% 사용자 입력 받기
user_input = input('각도를 입력하세요 (exit 입력 시 종료): ', 's');
% 종료 조건
if strcmpi(user_input, 'exit')
disp('프로그램을 종료합니다.');
break;
end
% 사용자 입력 실행
try
% 사용자 입력 실행 및 업데이트
eval(user_input); % 입력된 명령 실행
theta1 = deg2rad(theta1); % 각도를 라디안으로 변환
theta2 = deg2rad(theta2);
theta3 = deg2rad(theta3);
% 매니퓰레이터 업데이트
updateManipulator(theta1, theta2, theta3);
catch
disp('잘못된 입력입니다. 다시 입력하세요.');
end
end
%% 매니퓰레이터 업데이트 함수
function updateManipulator(theta1, theta2, theta3)
% 링크 길이
link_lengths = [36, 77.2, 100];
% 순기구학 계산 함수 정의 (지역 함수)
T_single = @(theta, d, a, alpha) [
cos(theta), -sin(theta)*cos(alpha), sin(theta)*sin(alpha), a*cos(theta);
sin(theta), cos(theta)*cos(alpha), -cos(theta)*sin(alpha), a*sin(theta);
0, sin(alpha), cos(alpha), d;
0, 0, 0, 1;
];
% DH 테이블 업데이트
DH_table = [
theta1, link_lengths(1), 0, pi/2;
```

```
theta2, 0, link_lengths(2), 0;
theta3, 0, link_lengths(3), 0;
];
% 순기구학 계산
T = eye(4); % 초기 변환 행렬
positions = [0, 0, 0]; % 기준 프레임의 원점
for i = 1:size(DH_table, 1)
theta = DH_table(i, 1);
d = DH_table(i, 2);
a = DH_table(i, 3);
alpha = DH_table(i, 4);
% 현재 조인트의 변환 행렬 계산
T_i = T_single(theta, d, a, alpha);
T = T * T_i; % 누적 변환 행렬
positions = [positions; T(1:3, 4)']; % 각 조인트 위치 저장
end
% 매니퓰레이터 그래프 플로팅
plot3(positions(:, 1), positions(:, 2), positions(:, 3), '-o', 'LineWidth', 2);
grid on;
xlabel('X (mm)');
ylabel('Y (mm)');
zlabel('Z (mm)');
title('Manipulator Forward Kinematics');
axis equal;
view(3); % 3D 뷰 설정
drawnow; % 그래프 업데이트
end
```

2-1. 순기구학 코드 설명

- DH 파라메터 테이블을 만들고 각도를 1개 변수로 잡은 후 동차 변환 행렬을 곱해나가는 방식을 코드로 구현했다.

- 이후 그래프로 해당 매니퓰레이터를 3차원에 이미지화 하는 방법을 채택하였다.

- 세부적인 코드 구성은 아래에서 다루겠다 .

```
while true
    user_input = input('각도를 입력하세요 (exit 입력 시 종료): ', 's');
    if strcmpi(user_input, 'exit')
        disp('프로그램을 종료합니다.');
        break;
    end
    try
        eval(user_input);
        theta1 = deg2rad(theta1);
        theta2 = deg2rad(theta2);
        theta3 = deg2rad(theta3);
        updateManipulator(theta1, theta2, theta3);
    catch
        disp('잘못된 입력입니다. 다시 입력하세요.');
    end
end
```

- 사용자에게 회전 각도를 입력받고 매니퓰레이터를 그래프로 표현함

```
DH_table = [
    theta1, link_lengths(1), 0,          pi/2;
    theta2, 0,                    link_lengths(2), 0;
    theta3, 0,                    link_lengths(3), 0;
];
```

- 입력받은 값을 DH 테이블로 표현함

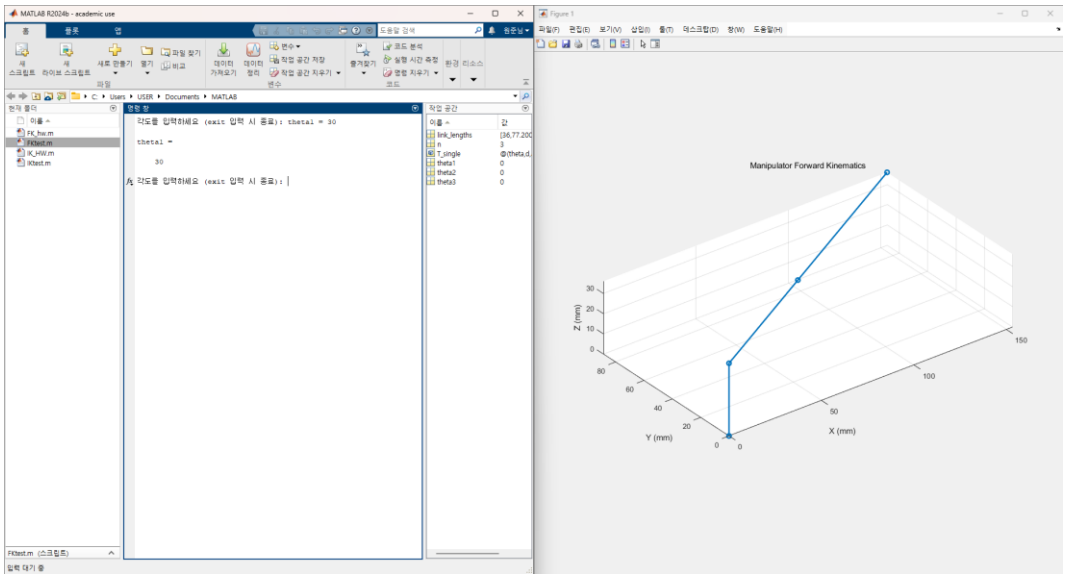
```
T = eye(4);
positions = [0, 0, 0];
```

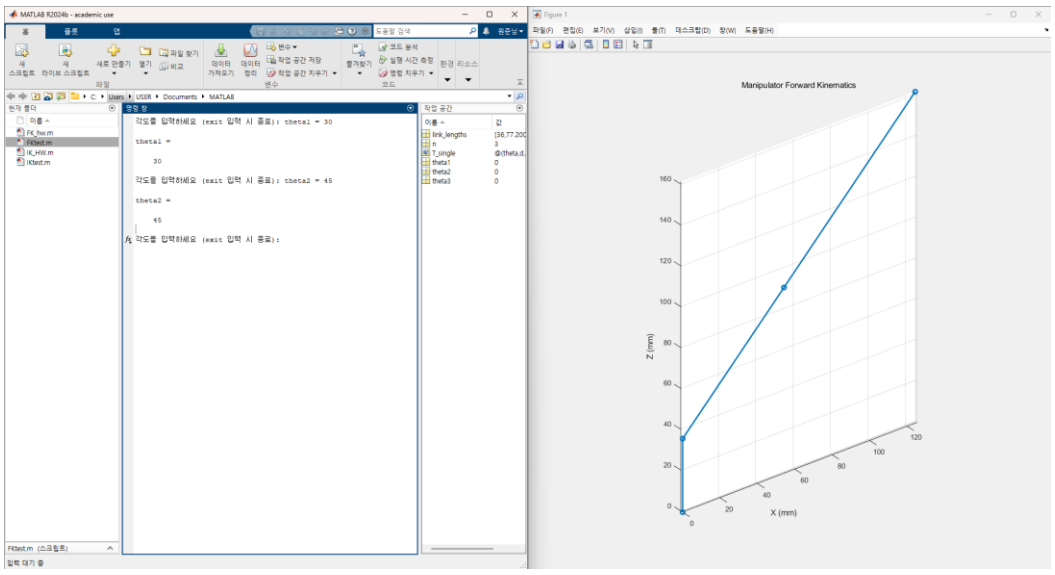
- T는 기준 좌표계에서의 변환행렬을 의미한다.
- Positions는 각 조인트의 위치를 저장할 행렬을 의미한다.

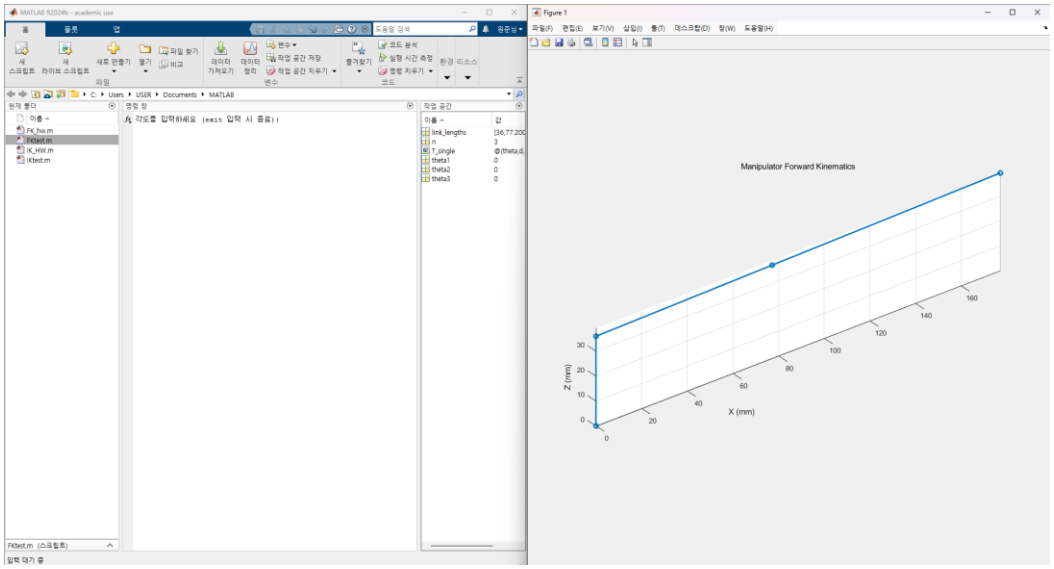
```
for i = 1:size(DH_table, 1)
    theta = DH_table(i, 1);
    d = DH_table(i, 2);
    a = DH_table(i, 3);
    alpha = DH_table(i, 4);

    T_i = T_single(theta, d, a, alpha);
    T = T * T_i;
    positions = [positions; T(1:3, 4)'];
end
```

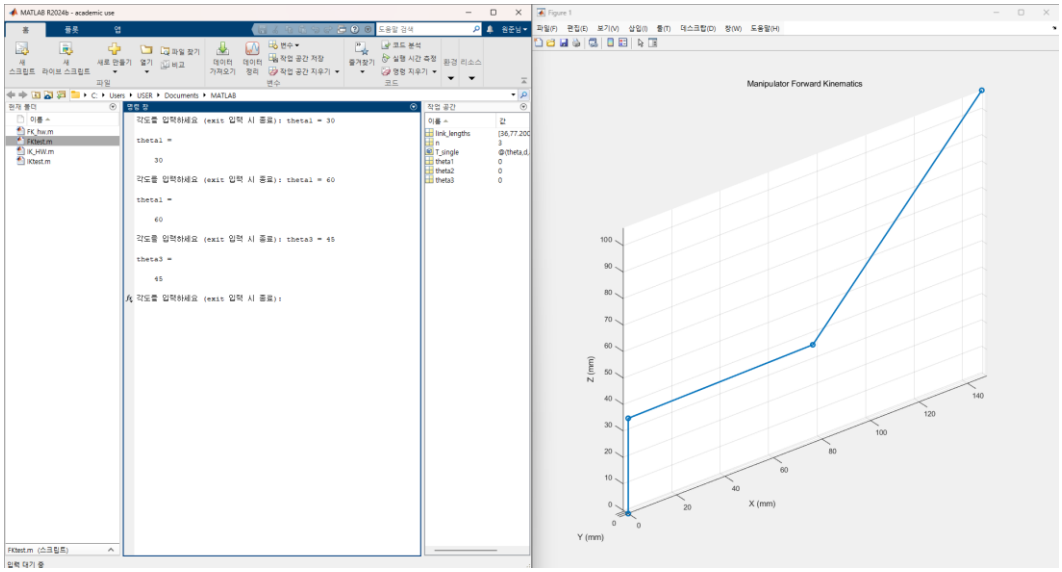
- 각 조인트의 변환 행렬(`T_i`)을 계산하고 누적하여 전체 매니퓰레이터의 위치를 계산한다.
- 이때 매 조인트의 위치를 positions에 좌표를 저장한다.

2-2. 실행 결과


각도를 입력하세요 (exit 입력 시 종료): theta1 = 30
theta1 =
30
각도를 입력하세요 (exit 입력 시 종료):
Manipulator Forward Kinematics
Z (mm)
X (mm)
Y (mm)


각도를 입력하세요 (exit 입력 시 종료): theta1 = 30
theta1 =
30
각도를 입력하세요 (exit 입력 시 종료): theta2 = 45
theta2 =
45
각도를 입력하세요 (exit 입력 시 종료):
Manipulator Forward Kinematics
Z (mm)
X (mm)


Manipulator Forward Kinematics
X (mm)
Z (mm)


각도를 입력하세요 (exit 입력 시 종료): theta1 = 30
theta1 =
30
각도를 입력하세요 (exit 입력 시 종료): theta1 = 60
theta1 =
60
각도를 입력하세요 (exit 입력 시 종료): theta3 = 45
theta3 =
45
각도를 입력하세요 (exit 입력 시 종료):
Manipulator Forward Kinematics
X (mm)
Y (mm)
Z (mm)

3. 역기구학 코드

```
clc;
clear;
close all;
%% 기본 설정
% 링크 길이 (mm 단위)
link_lengths = [36, 77.2, 100]; % 링크 1, 2, 3의 길이 (Base에서부터)
max_distance = sum(link_lengths(2:3)); % 링크 2와 링크 3의 최대 작업 거리
% 사용자로부터 목표 위치 입력받기
while true
try
% 목표 3D 평면 좌표 입력 (x, y, z)
user_input = input('목표 3D 위치 (x, y, z)를 입력하세요 [x y z]: ', 's');
if strcmpi(user_input, 'exit')
disp('프로그램을 종료합니다.');
break;
end
% 입력된 값을 숫자 배열로 변환
target = str2num(user_input); %#ok<ST2NM>
if numel(target) ~= 3
error('x, y, z의 3개의 값을 입력해야 합니다.');
end
x = target(1); % x 좌표
y = target(2); % y 좌표
z = target(3); % z 좌표
% Step 1: r 계산
r = sqrt(x^2 + y^2); % XY 평면 거리
if r == 0
error('x와 y는 동시에 0일 수 없습니다.');
end
% Step 2: Z 방향 보정
z_eff = z - link_lengths(1); % 링크 1 길이를 제외한 유효 z 값
% Step 3: 2D 역기구학 계산 (Roll 축)
[theta2, theta3] = inverseKinematics2D(r, z_eff, link_lengths(2:3), max_distance);
% Step 4: Roll 축 계산
theta1 = atan2(y, x); % 조인트 1의 회전각 (Yaw 축)
% 결과 출력
fprintf('조인트 각도 (degree):\n');
fprintf(' 조인트 1 (Yaw): %.2f°\n', rad2deg(theta1));
fprintf(' 조인트 2 (Roll): %.2f°\n', rad2deg(theta2));
fprintf(' 조인트 3 (Roll): %.2f°\n', rad2deg(theta3));
% 3D 매니퓰레이터 시각화
plotManipulator3D(theta1, theta2, theta3, link_lengths);
catch ME
disp(['오류: ', ME.message]);
end
end
%% 역기구학 계산 함수 (2D)
function [theta2, theta3] = inverseKinematics2D(r, z, link_lengths, max_distance)
% 링크 길이
link2 = link_lengths(1); % 링크 2의 길이
link3 = link_lengths(2); % 링크 3의 길이
% 목표 점까지의 직선 거리 계산
distance = sqrt(r^2 + z^2);
```

```
% 작업 공간 체크
if distance > max_distance
error('입력한 위치는 작업 공간 바깥에 있습니다.');
end
% 각도 계산
cos_theta3 = (r^2 + z^2 - (link2^2 + link3^2)) / (2 * link2 * link3);
cos_theta3 = max(min(cos_theta3, 1), -1); % 코사인 값 제한
theta3 = acos(cos_theta3); % 조인트 3 (Roll 축)
sin_theta3 = sqrt(1 - cos_theta3^2); % 삼각형의 높이 계산
theta2 = atan2(z, r) - atan2(link3 * sin_theta3, link2 + link3 * cos_theta3); % 조
인트 2 (Roll 축)
end
%% 3D 매니퓰레이터 시각화 함수
function plotManipulator3D(theta1, theta2, theta3, link_lengths)
% 링크 길이
l1 = link_lengths(1); % 링크 1의 길이
l2 = link_lengths(2); % 링크 2의 길이
l3 = link_lengths(3); % 링크 3의 길이
% 3D 좌표 계산
% 조인트 1 (Base에서 링크 1 끝)
x1 = 0;
y1 = 0;
z1 = l1;
% 조인트 2 위치 (링크 2 끝점, Roll 축 적용)
x2 = l2 * cos(theta1) * cos(theta2);
y2 = l2 * sin(theta1) * cos(theta2);
z2 = z1 + l2 * sin(theta2);
% 조인트 3 위치 (End Effector)
x3 = x2 + l3 * cos(theta1) * cos(theta2 + theta3);
y3 = y2 + l3 * sin(theta1) * cos(theta2 + theta3);
z3 = z2 + l3 * sin(theta2 + theta3);
% 3D 그래프 초기화
figure(1); % 3D 시각화를 표시할 창
clf; % 기존 그래프 초기화
hold on;
grid on;
% 3D 축 설정
axis equal;
view(3); % 3D 뷰 활성화
rotate3d on;
xlim([-200, 200]); % x축 범위
ylim([-200, 200]); % y축 범위
zlim([-200, 200]); % z축 범위
xlabel('X (mm)');
ylabel('Y (mm)');
zlabel('Z (mm)');
title('3D Manipulator Visualization');
% 매니퓰레이터 플롯
plot3([0, 0], [0, 0], [0, z1], 'k-o', 'LineWidth', 2); % 링크 1
plot3([0, x2], [0, y2], [z1, z2], 'b-o', 'LineWidth', 2); % 링크 2
plot3([x2, x3], [y2, y3], [z2, z3], 'r-o', 'LineWidth', 2); % 링크 3
% 조인트 및 끝점 표시
plot3(0, 0, 0, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k'); % Base
plot3(x2, y2, z2, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k'); % 조인트 2
plot3(x3, y3, z3, 'ko', 'MarkerSize', 8, 'MarkerFaceColor', 'k'); % End Effector
% 표시 업데이트
```

```
drawnow;
end
```

3-1. 역기구학 해석 코드 설명

- 역기구학에 대해 공부하면서 일반적인 풀이법은 없고 설계된 매니퓰레이터의 구조를 최대한 활용하는 방향으로 코드를 구성해야 한다고 느꼈다.

- 이전에 언급한 바와 같이 매트랩에서 해당 매니퓰레이터는 다음과 같이 구현할 것임을 밝힌 바가 있다.

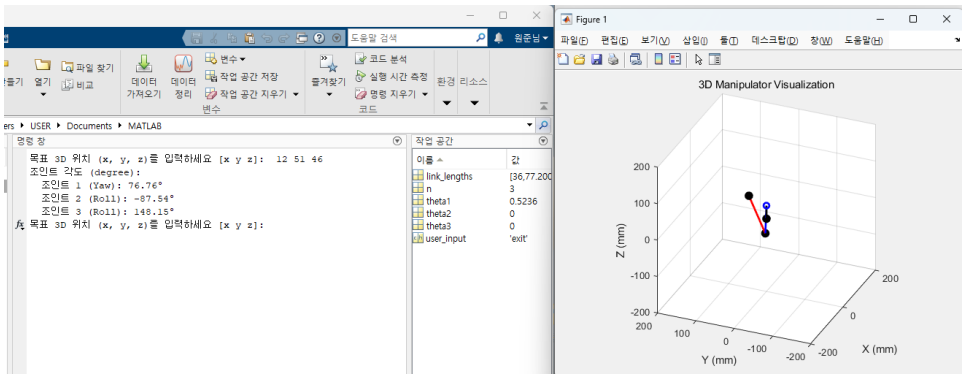
요 축 -> 링크 1번 (길이 36 mm) -> 롤 축 1번 -> 링크 2번 (길이 77.3mm) -> 롤 축 2번 -> 링크 3번 (100mm)

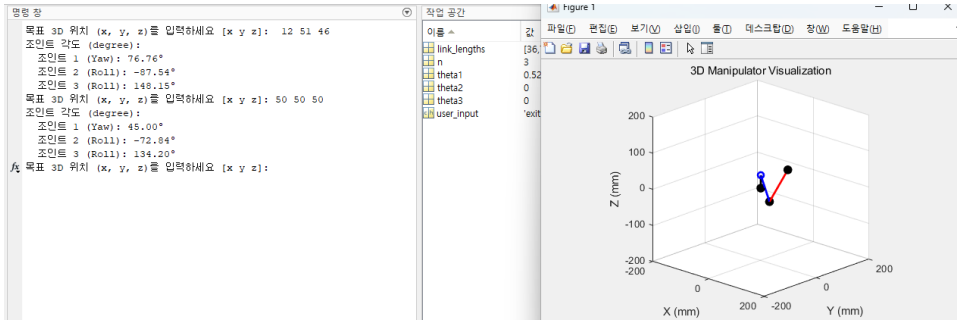
- 여기서 롤축 1번과 2번이 평행하고 링크 2번으로 연결된 점에서 롤축 1번부터 링크 3번까지가 평면에서 원 궤적을 그리며 움직인다는 것에서 아이디어를 얻어 롤축 1번 2번을 2차원에서 역기구학 해석을 하고 롤축 1번 2번의 각도를 결정한 후 요축 각도를 결정하는 식으로 코드를 구성하기로 하였다.

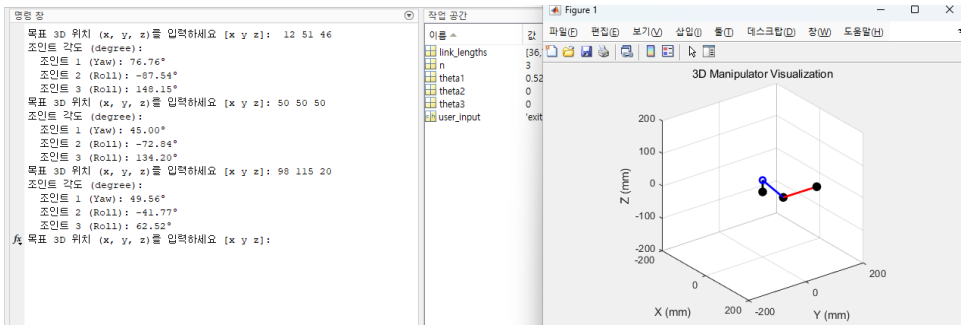
- 3차원의 매니퓰레이터의 두개의 축을 2차원에 옴겨서 해석하는 방법은 다음과 같다.

- 입력받는 좌표인 x, y, z의 x와 y에 피타고라스 정리를 적용해서 x-y평면상으로 목표 좌표를 선형 투사점과 원점의 거리 r을 잡는다. 그리고 요축의 조인트와 롤축 1번 조인트의 거리 36mm를 목표 좌표 z에 보정하여 새로운 변수 z_eff를 잡는다.

3-2. 실행 결과







- 실제 매니퓰레이터의 구조를 생각해보면 몸통의 위치나 팔의 구조적 한계에서 오는 워크스페이스의 제한이 있을 것인데 이부분을 구현하지 못한것이 아쉽다.