AI 로봇디자인 결과보고서

과목명 : AI 로봇디자인

담당교수 : 정성균 교수님, 봉재환 교수님

제출일 : 2022.06.18

학과 : 휴먼지능로봇공학과

3조(Chicken & Coke)

조원: 김동규, 김희찬, 서동민, 유동헌, 이상우, 전우혁

목차

1.전체 기구부 제작

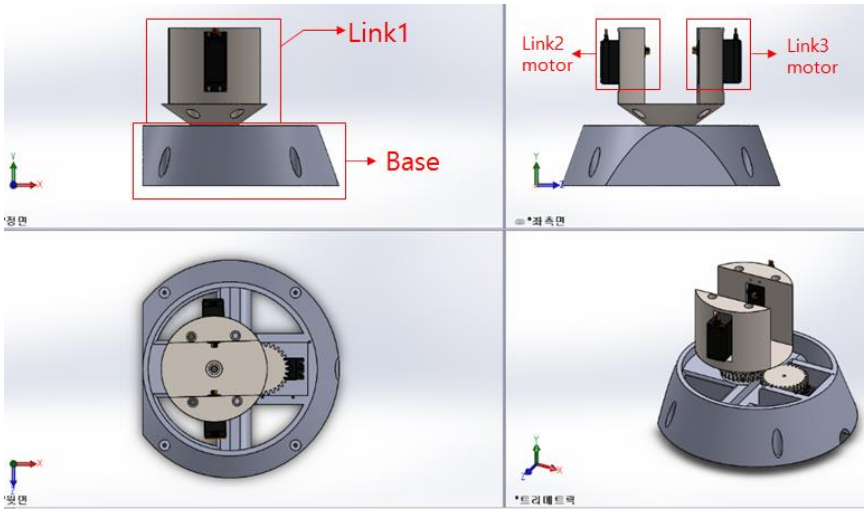

그림 1. 로봇의 베이스 및 링크 1

위 그림 1은 본 프로젝트 로봇의 베이스 부분이다. 베이스는 21cm 크기의 원형 스케치부터 원뿔대형태로 5.5cm까지 돌출되었다. 이와 같이 설계함으로써 링크의 움직임에 의한 실시간 무게 중심이 변화로 베이스가 넘어지지 않게 했다. 그리고 그림 1에서의 정면 사진에서 볼 수 있듯이 앞부분을 바닥면과 90° 를 이루는 면을 만들기 위해 돌출 컷을 진행했다. 이를 통해, 테스트 진행 시 베이스의 위치가 스타트 라인을 넘지 않으면서 링크 1 부품이 최대한 스타트 라인과 가까워지도록 했다. 베이스에는 Base Motor(motor1)을 체결했다. Base Motor는 전체 링크의 무게를 지탱하지 않기 위해 링크 1 부품에 직접 연결되지 않고 평기어에 의해 연결되었다. 기어는 링크 1과 기어 비 1:1비율로 연결 되어있다. 또한 링크 1에 해당되는 부품에 Link2 Motor, Link3 Motor이 모두 체결되어 있어 기구 전체의 회전 관성을 줄이고자 했다.

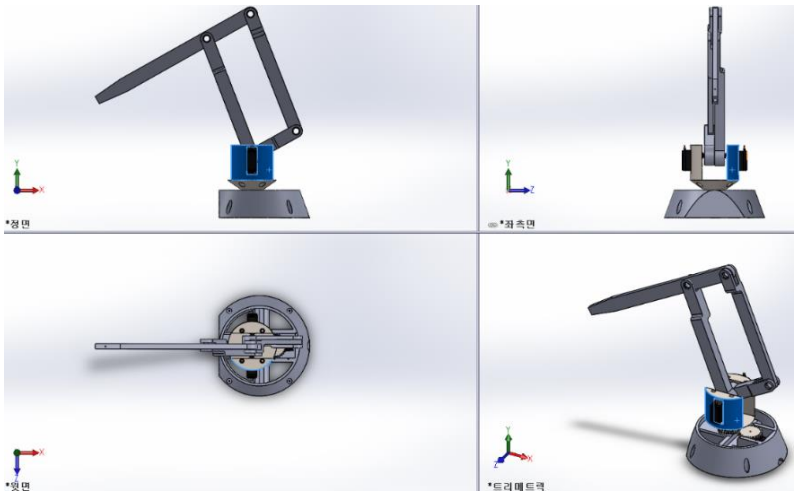

그림 2. 4절링크 형태의 Robot Arm

위 그림 2는 베이스와 연결된 링크 부분이다. 본 프로젝트에서는 회전 관성의 이득을 보면서, 타이밍벨트 메커니즘을 사용하지 않고 공학적 호기심을 해결하기 위해 다수의 링크를 사용하는 4절링크 메커니즘을 사용했다. Link 2 Motor와 Link 3 Motor는 링크와 직접 체결 되어있고, 모터가 연결되지 않은 관절은 내경이 6mm인 606z 볼 베어링이 들어가 있고 내경과 똑같은 규격의 M6 볼트, M6 나사, M6 와셔로 체결되었다. 여기서 L_1은 12.5cm, L_2는 25cm, L_3는 37cm로 정해서 로봇을 제작했다

2. 본 로봇 해석

위 본문 1과 같은 설계로 제작한 본 팀의 로봇을 3D 프린터를 이용해 베이스부터 각 링크들을 출력했다. 이렇게 출력한 각 부품과 3개의 모터를 조립했을 때 아래 그림 3과 같이 완성되었다. 그림 3에서도 볼 수 있듯이 본 팀의 로봇은 4절 링크 메커니즘을 기초로 설계 및 제작되었다.[2]

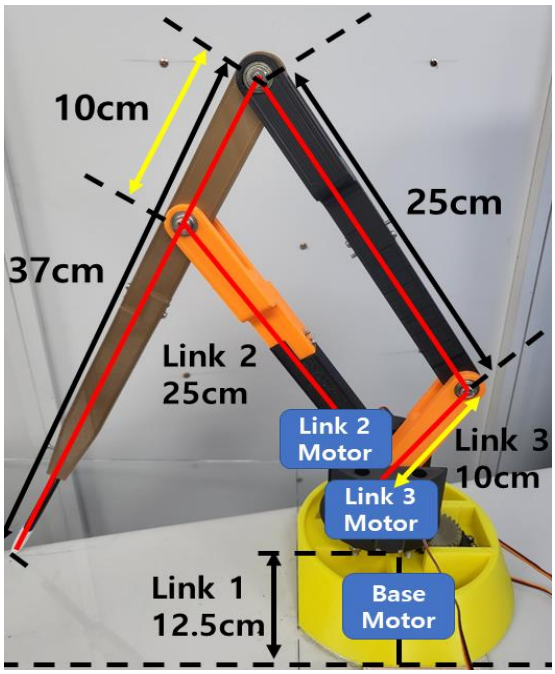

그림 3. 실제 제작된 로봇 및 로봇의 구성도

그에 따라 2번 링크의 길이와 3번 링크의 길이를 각각 25cm, 10cm이라고 설정했을 때, 위 각 링크들과 평행하며 동일한 길이의 링크를 추가함으로써 2번, 3번 링크의 각도 변화만큼 동일하게 4절 링크 부분이 변화되어 말단부(펜촉)가 위치하게 된다.

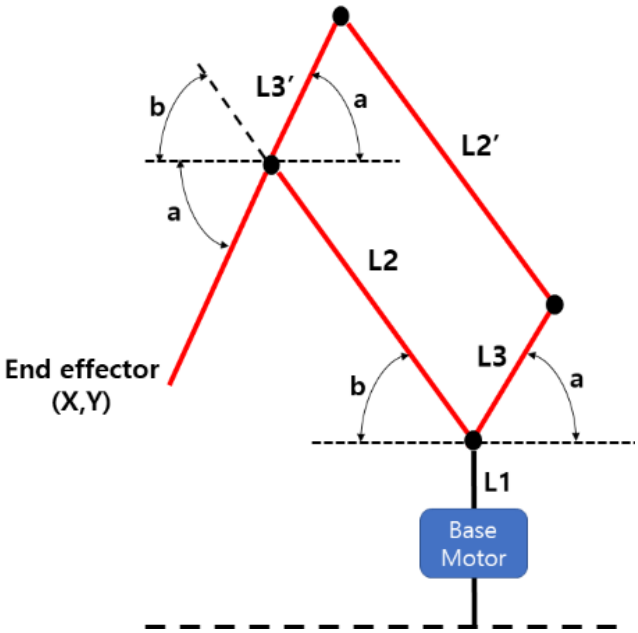

그림 4. 본 로봇 팔의 단순화된 구조 형태

3. 기구학 계산

3.1 정기구학(Forward Kinematics)

본 팀의 3DOF 로봇 팔의 DH Parameter는 아래 표 1과 같다.[3]

표 1 본 팀의 로봇 팔 DH Parameter

Joint	a	α	d	θ
1	0	90°	L_1	θ_1
2	L_2	0	0	θ_2
3	L_3	0	0	θ_3

로봇의 링크 길이는 각각 L_1은 12.5cm, L_2는 25cm, L_3는 37cm로 되어있기 때문에 이를 바탕으로 Forward Kinematics은 아래 수식 1을 이용하여 계산을 진행했다.

$$\begin{bmatrix} cos\theta & -sin\theta * cos\alpha & sin\theta * sin\alpha & L * cos\theta \\ sin\theta & cos\theta * cos\alpha & -cos\theta * sin\alpha & L * sin\theta \\ 0 & sin\alpha & cos\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

```
close all
clc

format short
syms th d  L  a
syms th1 d1 L1 a1
syms th2 d2 L2 th3

%베이스 모터1->모터2 사이 거리(l1)
d1=12.5;
%모터1->모터2 사이 거리(l2)
l2=25;
%모터2->엔드이펙터 사이 거리(l3)
l3=37;

%입력의 각도값
th=26;
th1=50;
th2=60;

T_01=HTM(th,d1,0,90);
T_12=HTM(th1,0,l2,0);
T_23=HTM(th2,0,l3,0);

[th]=armIK(30,0,0);

function T=HTM(th,d,L,alpha)

T=[cosd(th) -sind(th)*cosd(alpha) sind(th)*sind(alpha) L*cosd(th);
   sind(th) cosd(th)*cosd(alpha) -cosd(th)*sind(alpha) L*sind(th);
   0        sind(alpha)          cosd(alpha)                  d;
   0           0                    0                        1;];

end
```

그림 5. Forward Kinematics Matlab Code

Matlab에 그림 5과 같이 수식 1을 대입하여 Forward Kinematics의 ${}^{0}T_{3}$까지 식을 계산했다.

3.2 역기구학(Inverse kinematics)

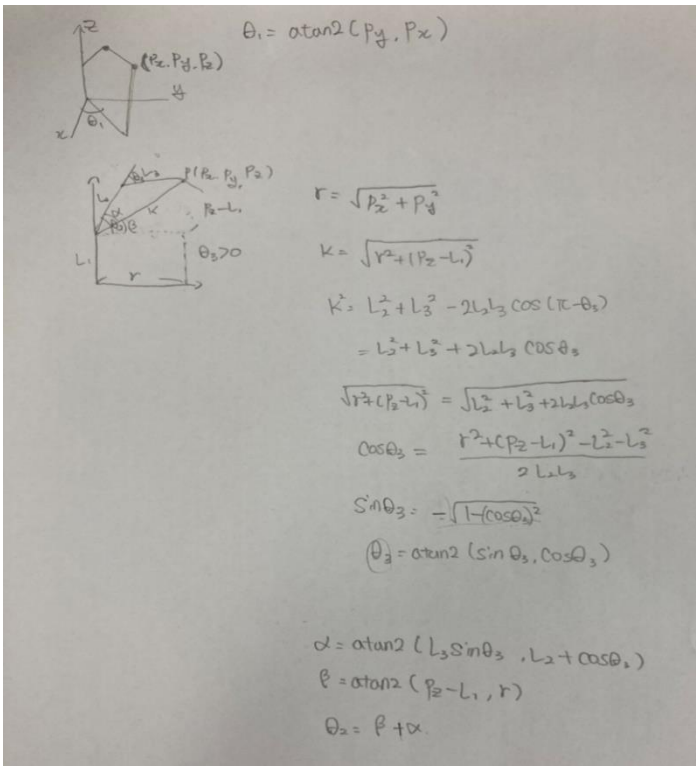

그림 6. Inverse Kinematics 풀이

그림 6과 같이 풀이된 Inverse kinematics은 그림 7과 같이 Matlab으로 구현했다. 또한 Inverse kinematics로 구한 θ값을 그림 5의 Forward kinematics Matlab 코드를 이용하여 좌표값을 비교하며 프로젝트를 진행했다.

```
close all
clc

format short
syms th d  L  a
syms th1 d1 L1 a1
syms th2 d2 L2 th3

function [th]=armIK(Px,Py,Pz)

%베이스 모터1->모터2 사이 거리(l1)
d1=12.5;

%모터1->모터2 사이 거리(l2)
l2=25;

%모터2->엔드이펙터 사이 거리(l3)
l3=37;

%베이스 바닥쪽 Px,Py 사이 거리
r=sqrt(Px^2+Py^2);

%역기구학을 통해 산출한 Inverse_th1~th3
%clac th1
Inv_th1=atan2d(Py,Px)

%clac th3
k=sqrt(r^2+(Pz-d1)^2)
Inv_th3_cos=(k^2-l2^2-l3^2)/(2*l2*l3)
Inv_th3_sin=-sqrt(1-(Inv_th3_cos^2))
Inv_th3=atan2d(Inv_th3_sin,Inv_th3_cos)

%clad th2
beta=atan2d(Pz-d1,r)

alpha=atan2d(l3*sind(Inv_th3),(l2+l3*cosd(Inv_th3)))

Inv_th2=beta-alpha


r_Inv_th1=90-Inv_th1;
r_Inv_th3=180-abs(Inv_th3+Inv_th2);
r_Inv_th2=180-Inv_th2;


th=[r_Inv_th1,r_Inv_th2,r_Inv_th3];
end
```

그림 7. Inverse Kinematics Matlab Code

앞서 구한 Inverse Kinematics를 통해 구해진 θ_2와 θ_3는 위 그림 4과 같이 단순화된 본 팀의 로봇 팔 구조에 따라 $\theta_2 = b$, $\theta_3 = a + b$ 같이 변경하여 설정했다. Link 2 Motor의 실제 구동 각도는 b이고, Link 3 Motor의 실제 구동 각도는 a이다.

4.카메라 보정 및 마커탐지

4.1 카메라 보정 기법

카메라의 보정기법은 chessboard 와 charucoboard 기법 두가지로 있다. 아래 그림 8과 같이 chessboard와 charucoboard의 사진 촬영은 높이를 3개로 나누어 촬영했다.

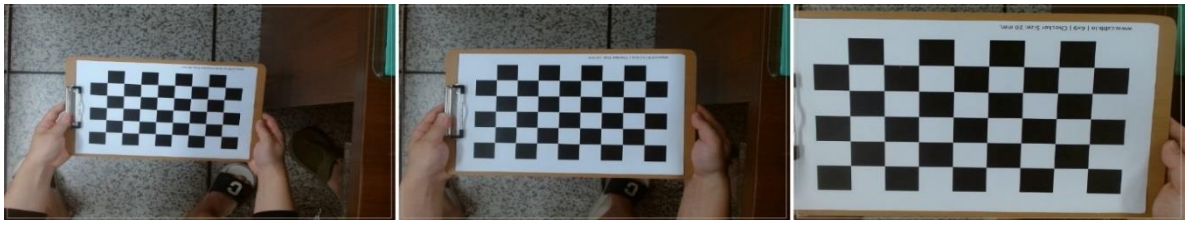

그림 8 높이에 변화에 따른 체스보드

아래 그림 9와 같이 각도별로 한 위치에서 정면, 상, 하, 좌, 우로 기울여서 촬영했다.

그림 9 각도에 변화에 따른 체스보드

아래 그림 10과 같이 위치별로 중앙으로부터 대각선 방향으로 이동하여 촬영했다.

그림 10 위치에 변화에 따른 체스보드

총 55장의 사진을 사용했으며 예시는 chessboard이나 charucoboard도 같은 방식으로 진행했다. charucoboard가 더 정확도가 높다고 알려져 있으나 본 팀은 실험을 통해서 두가지 기법을 모두 비교했을 때 chessboard가 더 높은 정확도를 보여 chessboard를 채택했다.

4.2 마커 인식 및 오차 확인

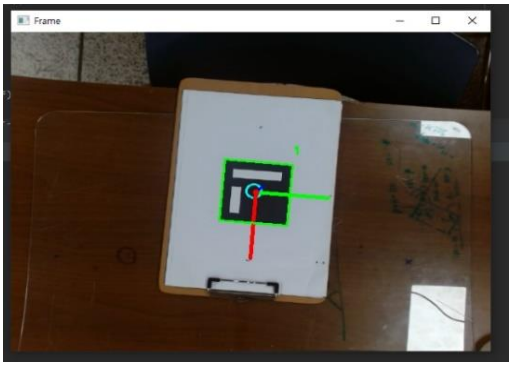

그림 11 마커 오차 확인

위 그림 11처럼 카메라 중심에 원을 그려서 Aruco 마커의 중심을 일치시켜 카메라상에서 가져온 tvec가 얼마나 오차가 생기는지 확인했다. 해당 오차는 tvec 기준, x축 약

1cm, y축 0.6cm, z축 약 2.5cm가 발생하는 것을 확인했다.

4.3 위치 오차 보정

교수님과의 미팅 후 본 팀은 오차를 줄이고 정확도를 향상시키는 방법으로 tvec, rvec 오차가 bias오차임을 확인하고 bias인 경우 오차를 해결하라는 조언을 받았다.

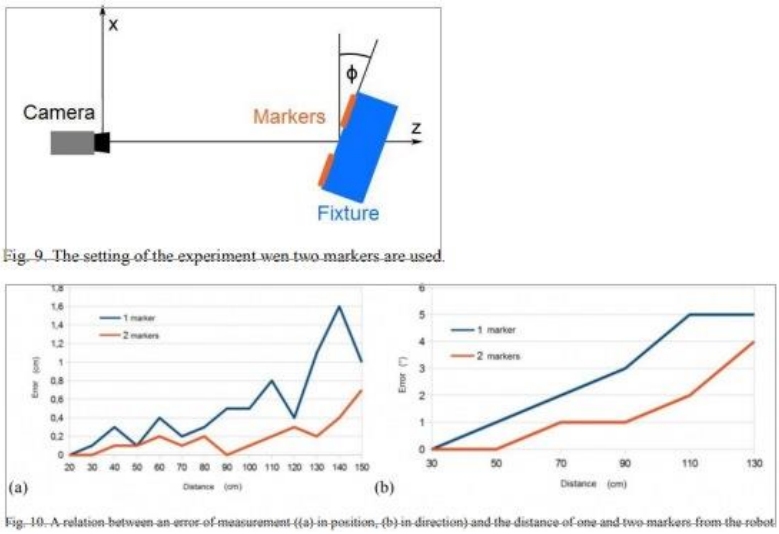

그림12. aruco마커 인식 시 거리에 따른 오차

위 그림 12은 동일한 카메라를 사용하여 두개의 aruco 마커 인식 시 거리에 따른 오차 차이를 나타낸다. [1]거리에 따른 다음 실험 결과를 통해 aruco 마커 인식 시 tvec, rvec 오차값을 최소화하기 위해 카메라 높이를 낮췄다. 이전 56cm에서 카메라 거치대와 링크 가동범위를 고려한 최소 높이 39cm로 변경했다. 이후 aruco마커를 이동 및 회전시키며 카메라 거치대의 고정이 잘 이루어졌는지 테스트를 진행했다.

5. 워크스페이스

5.1 워크스페이스 제작 및 이유

본 프로젝트에서 설계한 워크스페이스(workspace) 실물은 아래 그림 13과 같다.

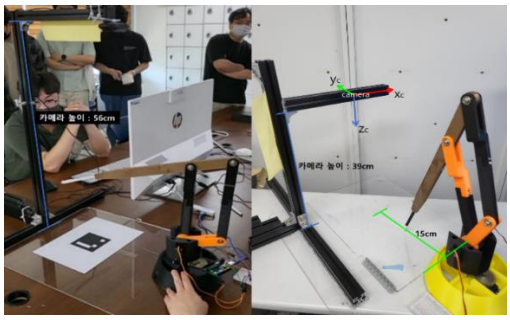

그림 13. 초기버전 및 최종 워크스페이스 제작

왼쪽 이미지의 초기 버전 워크스페이스는 카메라 거치대와 로봇 베이스가 고정되어 있지 않고 아크릴 판 위에 aruco 마커와 함께 올려둔 상태이다. 이후 카메라 중심좌표와 aruco마커 좌표와의 오차를 최대한 줄이고자 아크릴 판 위에 카메라 거치대와 로봇 베이스를 고정했다. 임의로 30x30 크기의 워크스페이스 공간을 설정하고 워크스페이스 중심으로 부터 15cm 떨어진 곳에 로봇 베이스를 고정했다. 카메라 좌표계와 aruco마커의 좌표계는 아래 그림 14과 같다.

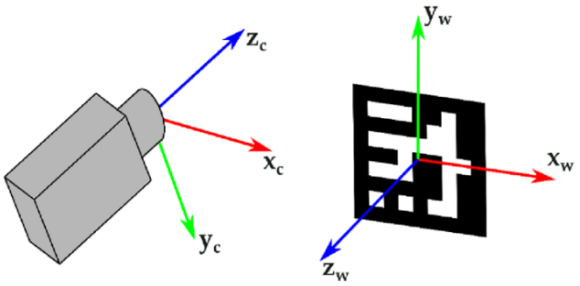

그림 14. 카메라 좌표계 및 aruco마커 좌표계

이때 카메라 거치대는 카메라 좌표계의 z축이 Base motor의 z축과 평행하도록 고정했다. 또 하나의 고려사항으로 카메라 좌표계의 x축 방향으로의 고정이 있다. 해당 사항은 aruco마커 인식 시 tvec, rvec 오차값을 최소화하기 위해 워크스페이스 중심에 4x4크기의 마커 인식을 진행 후 마커 중심과 카메라 중심을 일치시켰다.

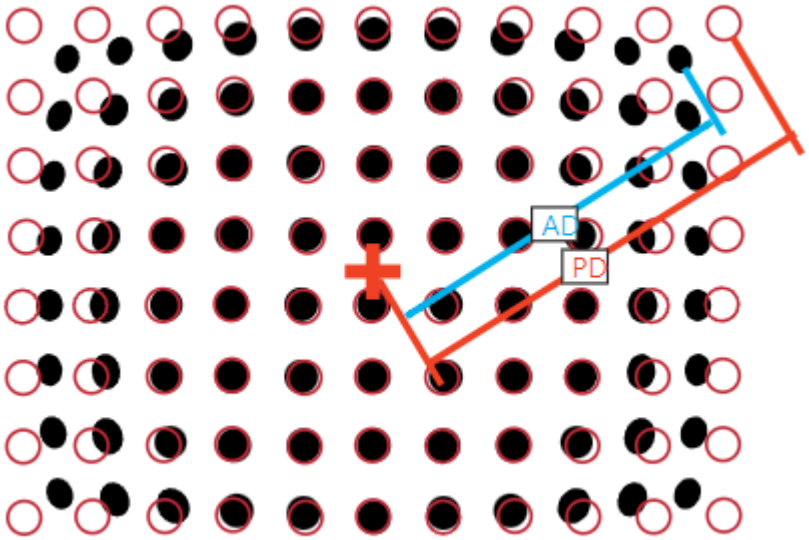

그림 15. 카메라 위치 선정 이유

위 그림 15과 같이 카메라의 중심에서 멀어질수록 왜곡이 크게 일어나기 때문에 위와 같이 마커 중심과 카메라 중심을 일치시켰다.

5.2 워크 스페이스 좌표계 변환

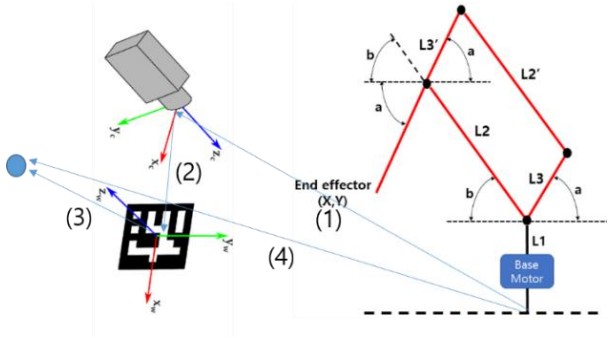

그림 16. 워크 스페이스 좌표계 변환

위 그림 16과 같이 지금 현재 로봇기준에서 마커의 좌표를 찍어야 하기에 지금까지 구해준 세개의 행렬을 계산했다. 그림 16 내 (1)은 베이스에서 캠까지의 행렬이며 우리가 설정하여 Known이다. (2)는 4번째 문단에서 다룬 마커에서 캠까지의 동차변환 행렬임으로 Known이다. (3)은 우리에게 주어지는 목표이며 Known이다. 이렇게 세개의 Known을 가지고 연산하여 베이스에서 목표까지의 동차변환 행렬인 (4)를 구해주었다. (4)의 행렬에서 X, Y, Z를 추출하여 본 팀에 맞는 역기구학식에 변환했다. 목표지점까지의 각도를 구하여 파이썬을 통해 아두이노로 전송했다.

6. 아두이노 모터제어

6.1 Python – Arduino 시리얼통신

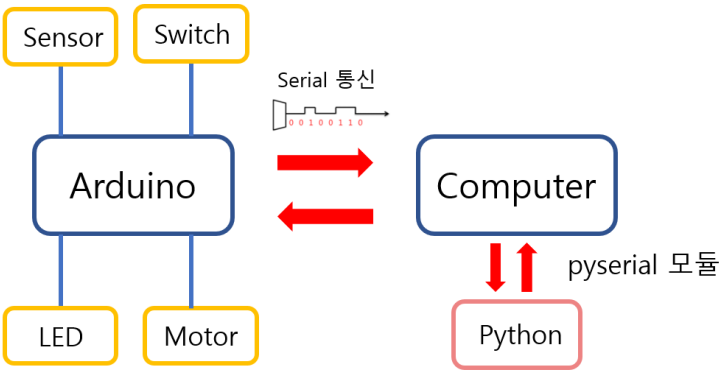

그림 17. 시리얼 통신 개념도

위 그림 17과 같이 아두이노는 컴퓨터와 통신을 하고, 파이썬은 pyserial 모듈을 이용해서 컴퓨터의 시리얼 통신 기능을 제어한다. 즉, 아두이노 IDE로 코딩을 해서 아두이노 보드에 시리얼 통신 기능이 포함된 코드를 컴파일하고, 별도로 파이썬으로 코딩을 했다.

Inverse kinematics 계산을 python에서 실시한 후 아두이노에 아래 그림 18과 같이 모터 각도를 string형식으로 전송한다.

```
theta = np.array([theta[0],  theta[1], theta[2]], dtype=str)
arduino.send_data(theta)
```

그림 18. Python to Arduino 각도 값 전송코드

입력 받은 각도 값을 아두이노에서는 배열로 저장하여 다른 변수나 추가적인 계산에 사용하게 된다. 아래 그림 19는 아두이노에서 배열로 저장하는 코드이다.

```
float value2 = Serial.parseFloat();
    for(int i=0;i<3;i++){
       if(val[i] == 0){
        val[i] = value2;
        value2 =0;
       }
       else{
        continue;
       }
     }
```

그림 19. 입력 값 배열저장코드

6.2 Servo motor calibration

모든 기구들을 출력 및 조립 후, 역기구학 코드와 연동하여 동작을 시켜보았다. 그 결과 역기구학 계산은 맞았지만, 로봇이 원하는 위치를 찍지 못하는 현상이 계속해서 발생했다. 교수님과의 미팅 후, 본 팀은 오차를 줄이고 정확도를 향상시키는 방법에 대하여 질문하였고 4가지의 방법을 제시 받았다. 그중 Servo motor calibration의 필요성을 인지했다.

각 모터(Base Motor, Link 2 Motor, Link 3 Motor) 와 수직이 되는 곳에 수평기를 이용하여 수평을 맞춘 후 각도기를 부착하여 모터가 입력한 값과 실제 돌아가는 각도가 얼마나 차이가 나는지를 확인했다. 결과는 다음 표와 같다.

입력각도	0°	10°	20°	30°	40°	50°	60°	70°	80°	90°
실제각도	-2°	8°	18°	25°	37°	45°	59°	67°	77°	87°
오차각도	-2°	-2°	-2°	-5°	-3°	-5°	-1°	-3°	-3°	-3°

입력각도	100°	110°	120°	130°	140°	150°	160°	170°	180°

실제각도	95°	105°	113°	121°	133°	142°	152°	160°	170°
오차각도	-5°	-5°	-7°	-9°	-7°	-8°	-8°	-10°	-10°

<표 2. Base Motor 각도측정 결과>

입력각도	0°	10°	20°	30°	40°	50°	60°	70°	80°	90°
실제각도	5°	15°	26°	36°	46°	56°	67°	79°	-	99°
오차각도	5°	5°	6°	6°	6°	6°	7°	9°	-	9°

입력각도	100°	110°	120°	130°	140°	150°	160°	170°
실제각도	-	120°	129°	138°	148°	157°	165°	175°
오차각도	-	10°	9°	8°	8°	7°	5°	5°

<표 3. Link 2 Motor 각도측정 결과>

입력각도	0°	10°	20°	30°	40°	50°	60°	70°	80°	90°
실제각도	-	8°	17°	27°	39°	49°	60°	71°	82°	95°
오차각도	-	-2°	-3°	-3°	-1°	-1°	0°	1°	2°	5°

입력각도	100°	110°	120°	130°	140°	150°	160°	170°
실제각도	-	117°	127°	137°	147°	157°	166°	176°
오차각도	-	7°	7°	7°	7°	7°	6°	6°

<표 4. Link 3 Motor 각도측정 결과>

위 표 2,3,4와 같은 결과를 얻어낸 후에 현실적으로 모터가 기구학적으로 움직일 수 있는 각도범위(노란색으로 칠해진 부분)를 지정하여 오차의 평균을 산출했다. Base Motor의 오차 평균값은 -4.63°, Link 2 Motor의 오차 평균값은 8.1°, Link 3 Motor의 오차 평균값은 -6.7°임을 확인했다.

아두이노에서 모터제어 코드는 각도 값을 입력해주면 그 위치로 바로 이동하는 형식이기 때문에 너무 빠르게 움직인다는 단점이 있다. 따라서 for문을 이용하여 입력 받은 각도까지 0.1°씩 줄어들거나 늘어나도록 하고 for문 내부에 delay를 0.005초씩 입력하여 입력 받은 각도까지 천천히 움직일 수 있도록 했다. 입력 받은 각도에 위에서 구한 모터들의 오차 평균값만큼 빼거나 더함으로써 모터가 정확한 각도에 위치하도록 했다.

6.3 motor 2개 각도 동시제어

앞에서 설명하였듯이 모터는 for 문을 통하여 천천히 목표 각도까지 이동하게 된다. Base Motor, Link 2 Motor, Link 3 Motor 순서로 모터가 천천히 움직이게 되는데, 바닥면 workspace를 찍을 때 수직인 workspace에 선을 그으며 이동하는 문제가 발생했다.

따라서 Base Motor를 제외한 나머지 모터를 동시에 제어하고자 하나의 for 문에 두 개의 각도 변수를 넣어서 한 번에 제어하는 방식으로 진행했다. 하지만 두 개의 모터가 받는 입력값의 차가 항상 같지는 않아서 하나의 모터만 목표 각도에 도달하고 나머지 모터는 목표 각도에 다다르지 않은 채로 작동이 종료되어버리는 문제가 발생한다. 해결 방법으로 목표 각도에 도달하지 못한 모터 값을 for 문 종료 시에 변수로 저장하고 이후에 저장한 값부터 목표값까지 다시 for 문을 통해서 그 모터만 돌려주는 방식으로 진행했다.

이후에 정확도를 증가시킬 수 있는 방법을 고민했다. 이후 모터 2개가 동시에 종료된다면 더욱 정확도가 상승하겠다는 예측을 하였고, 아래 그림 20, 21과 같이 Link 2 Motor의 초기값인 90, Link 3 Motor의 초기값인 130에서 입력값의 차이를 절대값으로 받아 큰 값과 작은 값으로 각각 분류하여 그 비율을 for문의 증감식 부분에 입력함과 동시에 2개의 모터가 종료되도록 했다.

```
dd1 = abs(90-val[1]);
dd2 = abs(130-val[2]);
df2 = 130-val[2];

if(dd1>dd2){
 dif_deg1 = (dd1/dd2)*0.1;
 dif_deg2 = 0.1;
}
else if(dd2>dd1){
 dif_deg1 = 0.1;
 dif_deg2 = (dd2/dd1)*0.1;
}
```

그림 20. 모터2개 비율계산코드

```
for(float v2=90,v3=130;v2<val[1],v3>val[2]+10;
    v2=v2+dif_deg1,v3=v3-dif_deg2)
```

그림 21. For문에 적용한 모습

6.4 본 원위치 회귀 방법

```
link3_motor.write(130);
delay(1000);
link2_motor.write(90);
delay(1000);
base_motor.write(180);
```

그림 22. 원위치 코드 1차

```
link3_motor.write(130);
link2_motor.write(90);
delay(1000);
base_motor.write(180);
```

그림 23. 원위치 코드 2차

실제로 시연하는 날 첫 번째 시도 직전에 점을 찍고 다시 원위치로 돌아오는 코드를 추가했다. 방법으로는 아두이노 초기 setup 값을 코드 마지막에 다시 추가하는 방법이었는데 실제로 시연 시에 문제가 발생했다. 원위치로 돌아오는 과정에서 Link 3 Motor, Link 2 Motor, Base Motor 순서로 하나씩 돌아오게 하였는데, 점을 잘 찍고 나서 돌아오는 과정에서 선을 그어버리면서 원위치 하게 되는 문제였다.

모든 팀이 시연을 종료하고 교수님께서 2차 기회를 주셔서 Link 2 Motor, Link 3 Motor를 동시에 돌리고 이후에 Base Motor를 돌리는 방법으로 코드를 수정하여 다시 한번 시연하였고, 선을 긋는 문제를 해결하여 1차보다 정확한 결과를 만들어 낼 수 있었다.

7. AI로봇디자인 보완점

본 팀에서 사용한 모터(MG996r)는 정확한 모터제어가 가능하지 않았다. 또한 backlash 가 예상보다 컸다. 분해능이 좀더 정확한 모터를 사용하면 정확도가 높아질 것으로 예상된다. 본 팀의 로봇 부품 중 기어의 backlash 가 발생했었다. 만약 Backlash 가 적절한 기어를 사용했다면 오차가 적었을 것으로 예상된다. 그리고 카메라 해상도에 따라 동차 변환 행렬 오류가 줄어든다.[1] 그림 24 는 카메라 해상도에 따른 오차를 나타낸 것이다.

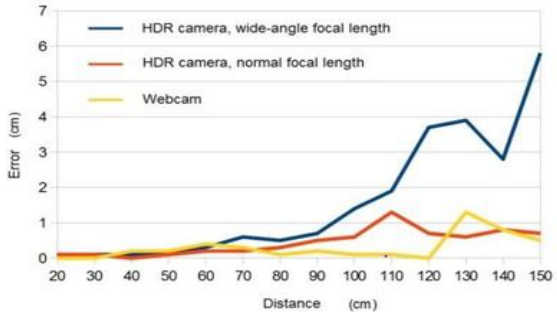

그림 24 카메라 해상도에 따른 오차

이처럼 만약 해상도가 높은 카메라를 사용했을 경우에 오차를 줄었을 것 같다. 마지막으로 배선 과정 중 모터 점퍼선에 납땜을 진행했다면 더 정확한 시그널을 보냈을 것으로 기대한다.

[Reference]

[1] Babinec, Andrej, et al. "Visual localization of mobile robot using artificial markers." Procedia Engineering 96 (2014): 1-9

[2] Suwarno, D.U.. (2016). Analysis kinematics graphically of the hastobot robotic ARM. 11. 6832-6835.

[3] M. M. U. Atique and M. A. R. Ahad, "Inverse Kinematics solution for a 3DOF robotic structure using Denavit-Hartenberg Convention," 2014 International Conference on Informatics, Electronics & Vision (ICIEV), 2014, pp. 1-5, doi: 10.1109/ICIEV.2014.6850854