1　目的

現在，産業用ロボットは，その精密な作業や人間が入ることのできない過酷な環境下での活躍，そして効率化など，工業の分野においてなくてはならない存在となっている．その中でも，ロボットアームは，見た目の動作が直感的であり，エンドエフェクタを取り替えることによりさまざまな作業ができることから，自動車産業をはじめとして幅広く使われている．

本実験では，6 自由度のロボットアームを使ってシミュレーションと実際の操作を行い，ロボット工学の基礎である 1. DH 法による座標系のとりかた 2. 同次変換行列 3. 順運動学 4. 逆運動学 について学ぶ．

2　実験装置と開発環境

本実験では，6 自由度の小型ロボットアーム，”myCobot 280 Pi” を用いる．これは，Raspberry Pi が組み込まれているため，OS を直接操作や ssh 接続して OS を遠隔操作するなどしてロボットを動かすことができる．

今回は自分の PC から ssh 接続して，OS を遠隔操作することによりロボットを操作した．なお，シミュレーションの実行環境を表 1 に示す．

表 1: 実行環境

OS	Ubutnu 24.04
Python の実行環境	Python 3.12.3 (venv)

3　実験

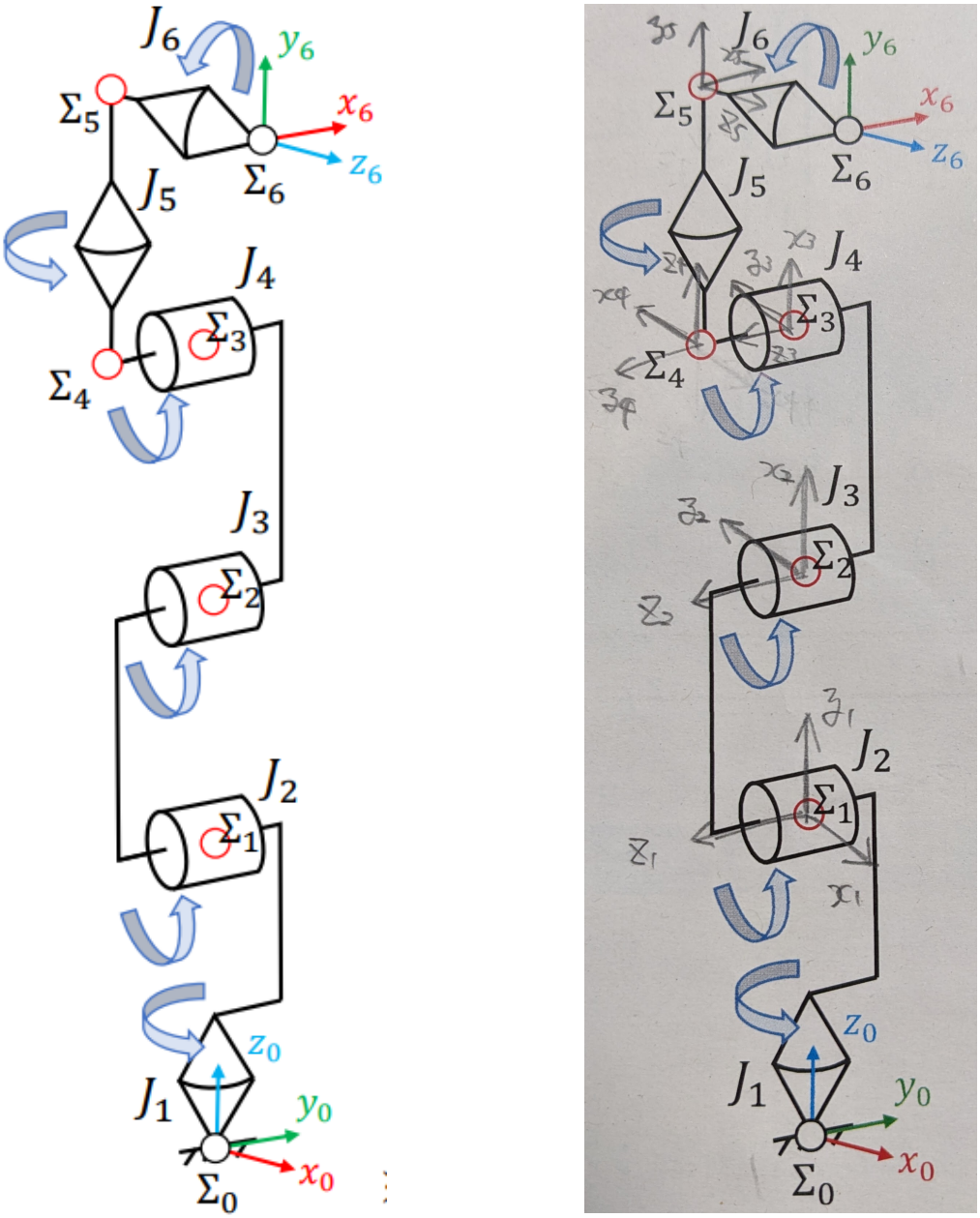
3.1　DH 法によりリンク間の座標 Σ_i を定義

図 1a に，直立状態のロボットアームのモデルを示す．まず，DH 法によりリンク間の座標を定義していく．図 1a より，関節はすべて回転関節であるから，z_i はすべて回転に対して右ねじの向きにとる．次に，隣接する関節どうしの z 軸の関係について，交錯する組と平行の組が表 2 のようにわけられる．

表 2: 隣接する関節どうしの z 軸の関係

交錯	$(z_0, z_1), (z_3, z_4), (z_4, z_5)$
平行	$(z_1, z_2), (z_2, z_3), (z_5, z_6)$

表 2 の関係に気をつけて座標 Σ_i を定義していくと，図 1b の形で座標系が定義できる．



(a) 直立状態

(b) DH 法により座標系を定義

図 1: ロボットアームのモデルと座標系

3.2 リンクパラメータを求め同次変換行列 ${}^{i-1}T_i$ を定義

座標系 Σ_{i-1} から Σ_i へ移動するときの関係を記述する同次変換行列 ${}^{i-1}T_i$ を定義する．$i-1$ から i への移動は次のように行う．

1. z_{i-1} 軸方向に d_i 並進

2. z_{i-1} 軸まわりに θ_i 回転
3. x_i 軸方向に l_i 並進
4. x_i 軸回りに α_i 回転

$i-1$ から i への移動をこのように定義した場合，同次変換行列 ${}^{i-1}T_i$ は式 (1) となる．

$$\begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & l_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & l_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

まずはロボットアームのリンクパラメータを求める．表 3 に示す．

表 3: リンクパラメータ

${}^{i-1}T_i$	d_i	θ_i	l_i	α_i
${}^{0}T_1$	d_1	θ_1	0	$\frac{\pi}{2}$
${}^{1}T_2$	0	θ_2	l_2	0
${}^{2}T_3$	0	θ_3	l_3	0
${}^{3}T_4$	d_4	θ_4	0	$\frac{\pi}{2}$
${}^{4}T_5$	d_5	θ_5	0	$\frac{\pi}{2}$
${}^{5}T_6$	d_6	θ_6	0	0

表 3 を式 (1) に代入し，${}^{i-1}T_i$ を求めた．式 (2)〜(7) に示す．ただし，$C_i \coloneqq \cos\theta_i, S_i \coloneqq \sin\theta_i$ である．

$$
{}^{0}T_1 = \begin{bmatrix} C_1 & 0 & S_1 & 0 \\ S_1 & 0 & -C_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}
$$

$$
{}^{1}T_2 = \begin{bmatrix} C_2 & -S_2 & 0 & l_2C_2 \\ S_2 & C_2 & 0 & l_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}
$$

$$
{}^{2}T_3 = \begin{bmatrix} C_3 & -S_3 & 0 & l_3C_3 \\ S_3 & C_3 & 0 & l_3S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}
$$

$$
{}^{3}T_4 = \begin{bmatrix} C_4 & 0 & S_4 & 0 \\ S_4 & 0 & -C_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 4 \end{bmatrix} \tag{5}
$$

$$
{}^{4}T_5 = \begin{bmatrix} C_5 & 0 & S_5 & 0 \\ S_5 & 0 & -C_5 & 0 \\ 0 & 1 & 0 & d_5 \\ 0 & 0 & 0 & 5 \end{bmatrix} \tag{6}
$$

$$
{}^{5}T_6 = \begin{bmatrix} C_6 & -S_6 & 0 & 0 \\ S_6 & C_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}
$$

3.3 DH 法における角度 θ_i とサーボモータの角度 J_i の対応付け

DH 法における角度 θ_i は x_{i-1} 軸と x_i 軸のなす角であるのに対し，ロボットの関節に搭載されているサーボモータの角度 J_i は図 1 の直立姿勢において $J_i = 0, i = 1 \ldots 6$ である．また，θ_i が弧度法なのに対し，J_i は分度法である．この異なる定義の角度 θ_i と J_i の対応付けを行った．式 (8)〜(13) に示す．

$$
\theta_1 = \frac{J_1}{180}\pi + 0 \tag{8}
$$

$$
\theta_2 = \frac{J_2}{180}\pi + \frac{\pi}{2} \tag{9}
$$

$$
\theta_3 = \frac{J_3}{180}\pi + 0 \tag{10}
$$

$$
\theta_4 = \frac{J_4}{180}\pi + \frac{\pi}{2} \tag{11}
$$

$$
\theta_5 = \frac{J_5}{180}\pi - \frac{\pi}{2} \tag{12}
$$

$$
\theta_6 = \frac{J_6}{180}\pi + 0 \tag{13}
$$

3.4　座標系 Σ_0 基準でのアーム手先位置を求める順運動学問題

手先の位置は，Σ_6 基準では ${}^6p_t = \begin{bmatrix}0 & 0 & 0 & 1\end{bmatrix}^\mathsf{T}$ である．これに対して式 (2)〜(7) の同次変換行列を左から順にかけていくことにより，Σ_0 基準の手先位置 ${}^0p_t$ が式 (16) のように求まる．導出過程を式 (14)〜(16) に示す．なお，式 (14) の行列の積の展開には Sympy を用いた．

$$
{}^0p_t = {}^0T_1^1T_2^2T_3^3T_4^4T_5^5T_6^6p_t \tag{14}
$$

$$
= \begin{bmatrix}
C_1C_2C_3l_3 - C_1S_2S_3 + C_1C_2l_2 + C_1d_5(C_4S_{23} + S_4C_{23}) + S_1d_4 - d_6\{-C_1S_5(C_4C_{23} - S_4S_{23}) + S_1C_5\} \\
S_1C_2C_3l_3 - S_1S_2S_3 + S_1C_2l_2 + C_1d_5(C_4S_{23} + S_4C_{23}) - C_1d_4 - d_6\{S_1S_5(C_4C_{23} - S_4S_{23}) + C_1C_5\} \\
C_2S_3l_3 + S_2C_3l_3 + S_5d_6(C_4S_{23} + S_4C_{23}) + d_1 - d_5(C_4S_{23} - S_4S_{23}) \\
1
\end{bmatrix} \tag{15}
$$

$$
= \begin{bmatrix}
(d_5S_4 + d_6C_4S_5)C_1C_{23} + (d_5C_4 - d_6S_4S_5)C_1S_{23} + (l_2 + l_3C_3)C_1C_2 + (d_4 - d_6C_5)S_1 - l_3C_1S_2S_3 \\
(d_5S_4 + d_6C_4S_5)S_1S_{23} + (d_5C_4 - d_6S_4S_5)S_1S_{23} + (l_2 + l_3C_3)S_1C_2 + (d_4 - d_6C_5)C_1 - l_3S_1S_2S_3 \\
(d_5S_4 + d_6C_4S_5)S_{23} + (d_5C_4 - d_6S_4S_5)C_{23} + (l_2 + l_3C_3)S_2 + d_1 + l_3C_2S_3 \\
1
\end{bmatrix} \tag{16}
$$

3.5　6 関節角度を入力し，全関節を同時に動作させるプログラムの作成

sample.py を書き換えることにより，6 関節角度を入力し，全関節を同時に動作させるプログラムを作成した．実際に作成したプログラムは付録の Listing 2 に掲載しておく．

6 関節角度に $\boldsymbol{J} = \begin{bmatrix}-12.08 & -51.17 & -123.8 & 84.96 & 0.0 & -12.08\end{bmatrix}^\mathsf{T}$ を与えたときのシミュレーション結果を図 2 に示す．

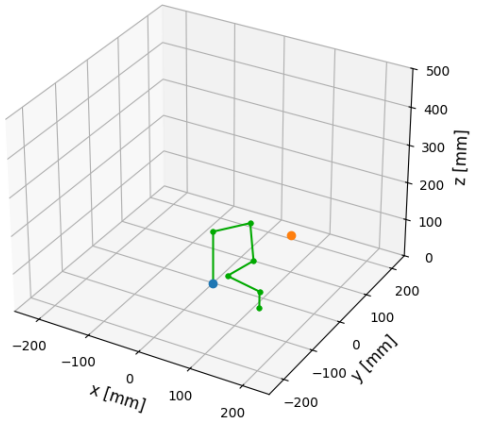


図 2: 6 関節角度に $\boldsymbol{J} = \begin{bmatrix}-12.08 & -51.17 & -123.8 & 84.96 & 0.0 & -12.08\end{bmatrix}^\mathsf{T}$ を与えたときのロボットの形状

3.6　順運動学により，関節角度から手先位置を計算するプログラムの作成

式 (16) に示した順運動学の式を使い，関節角度から手先位置を計算するプログラムを作成した．実際に作成したプログラムは付録の Listing 3 に掲載しておく．

次の関節角度と手先位置の組の例について，関節角度をシミュレータに入力し，手先位置の同次座標が正しく出力されること，マーカ位置と手先位置が一致することを確認した．コマンドラインの出力については以下に，シミュレーション結果の画像出力については図 3 に示す．

(a) $\boldsymbol{J} = \begin{bmatrix} -12.08 & -51.17 & -123.8 & 84.96 & 0.0 & -12.08 \end{bmatrix}^{\mathsf{T}}, {}^{0}p_t = \begin{bmatrix} 150 & -100 & 70 & 1 \end{bmatrix}^{\mathsf{T}}$
出力された同次座標

```
[149.98707521 -99.99309332  69.97903252   1.        ]
```

(b) $\boldsymbol{J} = \begin{bmatrix} -34.7 & -63.36 & -119.18 & 92.54 & 0.0 & -34.7 \end{bmatrix}^{\mathsf{T}}, {}^{0}p_t = \begin{bmatrix} 100 & -150 & 50 & 1 \end{bmatrix}^{\mathsf{T}}$
出力された同次座標

```
[ 100.00154038 -149.99662236   49.99582456    1.        ]
```

(c) $\boldsymbol{J} = \begin{bmatrix} 26.27 & -26.46 & -146.28 & 82.74 & 0.0 & 26.27 \end{bmatrix}^{\mathsf{T}}, {}^{0}p_t = \begin{bmatrix} 150 & 0 & 100 & 1 \end{bmatrix}^{\mathsf{T}}$
出力された同次座標

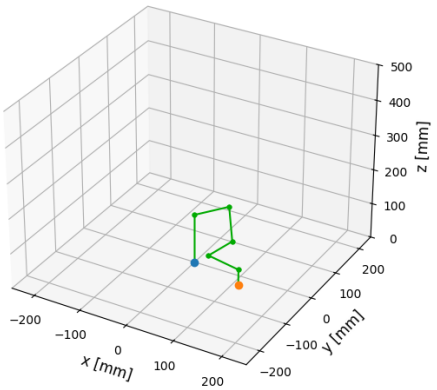
```
[ 1.49995606e+02 -1.87934829e-03  1.00004760e+02  1.00000000e+00]
```

(d) $\boldsymbol{J} = \begin{bmatrix} 55.30 & -51.17 & -123.80 & 84.96 & 0.0 & 55.30 \end{bmatrix}^{\mathsf{T}}, {}^{0}p_t = \begin{bmatrix} 150 & 100 & 70 & 1 \end{bmatrix}^{\mathsf{T}}$
出力された同次座標

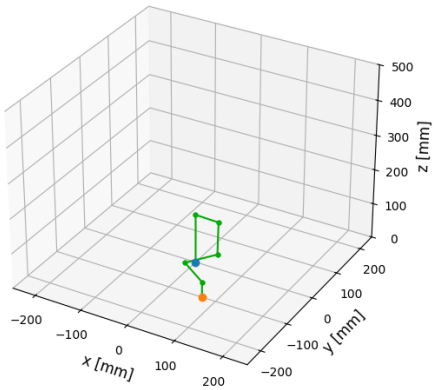
```
[149.98888922  99.9903723   69.97903252   1.        ]
```

(e) $\boldsymbol{J} = \begin{bmatrix} 77.92 & -63.36 & -119.18 & 92.54 & 0.0 & 77.92 \end{bmatrix}^{\mathsf{T}}, {}^{0}p_t = \begin{bmatrix} 100 & 150 & 50 & 1 \end{bmatrix}^{\mathsf{T}}$
出力された同次座標

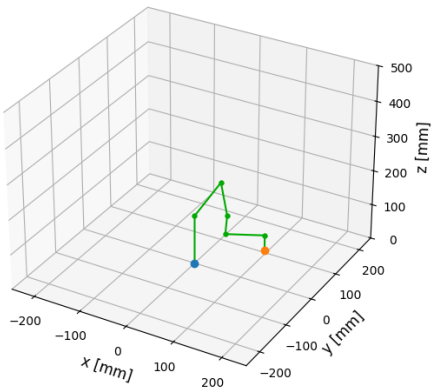
```
[ 99.99593616 150.0003585   49.99582456   1.        ]
```



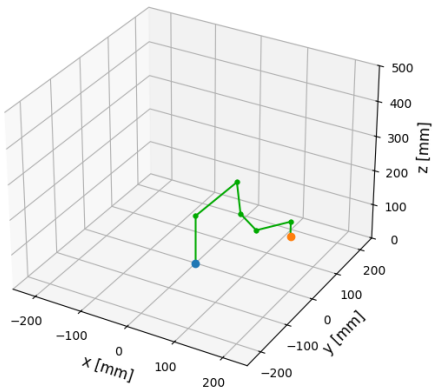
(a)



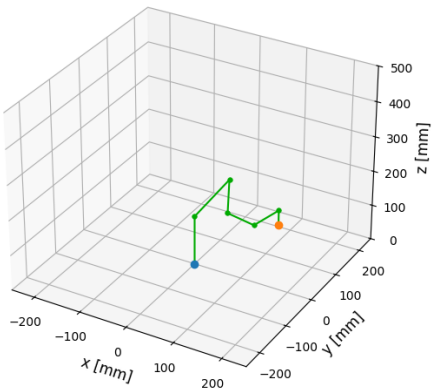
(b)



(c)



(d)



(e)

図 3: 順運動学から計算した手先位置

3.7 Z 方向の手先位置が 15 mm 以内の場合にエラーを発生させるプログラム

3.6 節に対して，Z 方向の手先位置が 15 mm 以内の場合に Z_ERROR を発生させるようにした．実際に作成したプログラムは付録の Listing 4 に掲載しておく．

次の関節角度と手先位置の組の例について，関節角度をシミュレータに入力し，(a) については正しく同次座標が求まること，(b)-(d) についてはエラーが出ることを確認した．その結果を以下に示す．

(a) $\boldsymbol{J} = \begin{bmatrix} 63.24 & -41.72 & -103.45 & 55.17 & 0.0 & 63.24 \end{bmatrix}^{\mathsf{T}}, {}^{0}p_t = \begin{bmatrix} 150 & 150 & 100 & 1 \end{bmatrix}^{\mathsf{T}}$
出力された同次座標

```
[149.99687637 150.00915832 100.00158833   1.        ]^\mathsf{T}
```

(b) $\boldsymbol{J} = \begin{bmatrix} 63.24 & -85.92 & -83.17 & 79.09 & 0.0 & 63.24 \end{bmatrix}^{\mathsf{T}}, {}^{0}p_t = \begin{bmatrix} 150 & 150 & 10 & 1 \end{bmatrix}^{\mathsf{T}}$
出力されたエラー

```
ZError: z error
```

(c) $\boldsymbol{J} = \begin{bmatrix} 145.3 & -63.36 & -119.18 & 92.54 & 0.0 & 145.3 \end{bmatrix}^{\mathsf{T}}, {}^{0}p_t = \begin{bmatrix} -100 & 150 & 50 & 1 \end{bmatrix}^{\mathsf{T}}$
出力されたエラー

```
AngleError: J1 angle error
```

(d) $\boldsymbol{J} = \begin{bmatrix} 41.6 & -118.04 & -155.24 & 183.28 & 0.0 & 41.6 \end{bmatrix}^{\mathsf{T}}, {}^{0}p_t = \begin{bmatrix} 100 & 50 & 50 & 1 \end{bmatrix}^{\mathsf{T}}$
出力されたエラー

```
AngleError: J3 angle error
```

3.8 実機で 3.7 節のプログラムの動作を確認

実機で 3.7 節のプログラム (Listing 4) の動作を確認した．

3.6 節の角度例を入力すると，実機においてもシミュレータと同じ手先位置・形状になることが確認できた．また，3.7 節の角度例を入力すると，実機においても同じエラーが出力され，エラーの場合はロボットが動作しないことを確認できた．

3.9　関節角度間の関係に拘束条件を与えた場合の順運動学

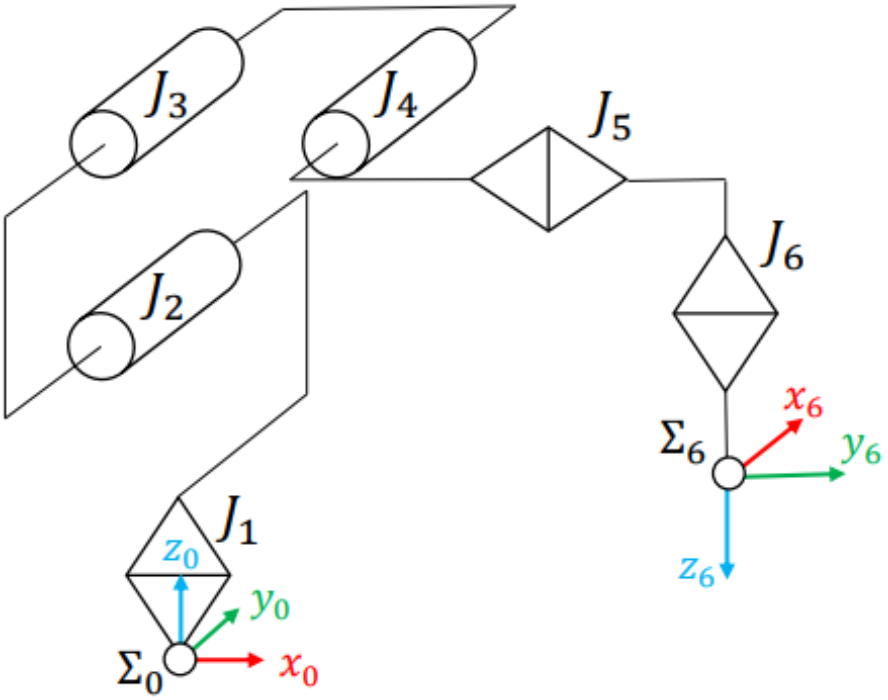


図 4: 関節角度間の拘束条件

図 4 に示すような拘束条件を与え，順運動学を簡略化する．図 4 の拘束条件を表す式は式 (17)〜(19) である．

$$\theta_4 = \frac{\pi}{2} - \theta_2 - \theta_3 \tag{17}$$

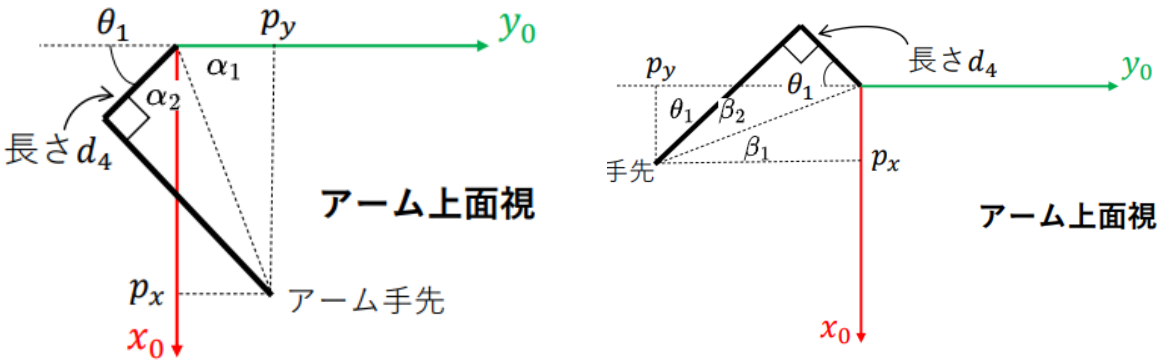
$$\theta_5 = -\frac{\pi}{2} \tag{18}$$

$$\theta_6 = \theta_1 \tag{19}$$

式 (17)〜(19) の条件を与えたとき，式 (16) に示した手先位置は式 (20) となる．

$${}^0p_t = \begin{bmatrix} l_3C_1C_2C_3 - l_3C_1S_2S_3 + l_2C_1C_2 + d_4S_1 + d_5C_1 \\ l_3S_1C_2C_3 - l_3S_1S_2S_3 + l_2S_1C_2 - d_4C_1 + d_5S_1 \\ l_3C_2S_3 + l_3S_2C_3 + l_2S_2 + d_1 - d_6 \\ 1 \end{bmatrix} \tag{20}$$

3.10　手先位置が与えられたときの θ_1 を求める



(a) $p_y > 0$　　**(b)** $p_y \leq 0$

図 5: p_y で場合分けした θ_1

手先位置 $\begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^\mathsf{T}$ が与えられたときの θ_1 を求める．p_y で場合分けすると，図 5 のようになる．それぞれの場合について θ_1 を求めていく．

まず，$p_y > 0$ の場合について考える．図 5a のように角度 α_1, α_2 を定義する．α_1, α_2 はそれぞれ式 (21), (22) となる．

$$\alpha_1 = \mathrm{atan2}(p_x, p_y) \tag{21}$$

$$\alpha_2 = \cos^{-1}\left(\frac{d_4}{\sqrt{p_x^2 + p_y^2}}\right) \tag{22}$$

よって，$\theta_1 (p_y > 0)$ が式 (24) のように求まる．

$$\theta_1 = \pi - \alpha_1 - \alpha_2 \tag{23}$$

$$= \pi - \mathrm{atan2}(p_x, p_y) - \cos^{-1}\left(\frac{d_4}{\sqrt{p_x^2 + p_y^2}}\right) \tag{24}$$

次に，$p_y \leq 0$ の場合について考える．図 5b のように角度 β_1, β_2 を定義する．$p_y \leq 0$ であることに注意すると，β_1, β_2 はそれぞれ式 (25), (26) となる．

$$\beta_1 = \sin^{-1}\left(\frac{d_4}{\sqrt{p_x^2 + p_y^2}}\right) \tag{25}$$

$$\beta_2 = \mathrm{atan2}(-p_y, p_x) \tag{26}$$

$$\tag{27}$$

よって，$\theta_1(p_y \leq 0)$ が式 (29) のように求まる．

$$\theta_1 = \frac{\pi}{2} - \beta_1 - \beta_2 \tag{28}$$

$$= \frac{\pi}{2} - \mathrm{atan2}(-p_y, p_x) - \sin^{-1}\left(\frac{d_4}{\sqrt{p_x^2 + p_y^2}}\right) \tag{29}$$

したがって，式 (24), (29) より，θ_1 は式 (30) のように求まる．

$$\theta_1 = \begin{cases} \pi - \mathrm{atan2}(p_x, p_y) - \cos^{-1}\left(\dfrac{d_4}{\sqrt{p_x^2 + p_y^2}}\right) & \text{if} p_y > 0 \\ \dfrac{\pi}{2} - \mathrm{atan2}(-p_y, p_x) - \sin^{-1}\left(\dfrac{d_4}{\sqrt{p_x^2 + p_y^2}}\right) & \text{if} p_y \leq 0 \end{cases} \tag{30}$$

3.11 手先位置が与えられたときの θ_2, θ_3 を求める

手先位置 $\begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^\mathsf{T}$ が与えられたときの θ_2, θ_3 を求める．
式 (20) より，

$$\begin{bmatrix} p_x \\ p_z \end{bmatrix} = \begin{bmatrix} l_3 C_1 C_3 + l_2 C_1 & -l_3 C_1 S_3 \\ l_3 S_3 & l_3 C_3 + l_2 \end{bmatrix} \begin{bmatrix} C_2 \\ S_2 \end{bmatrix} + \begin{bmatrix} d_5 C_1 + d_4 S_1 \\ d_1 - d_6 \end{bmatrix} \tag{31}$$

$$\therefore \begin{bmatrix} \dfrac{p_x - d_4 S_1 - d_5 C_1}{C_1} \\ p_z - d_1 + d_6 \end{bmatrix} = \begin{bmatrix} l_3 C_3 + l_2 & -l_3 S_3 \\ l_3 S_3 & l_3 C_3 + l_2 \end{bmatrix} \begin{bmatrix} C_2 \\ S_2 \end{bmatrix} \tag{32}$$

式 (32) の左辺を極座標形式で表現する．

$$\begin{bmatrix} p'_x \\ p'_z \end{bmatrix} \coloneqq \begin{bmatrix} \dfrac{p_x - d_4 S_1 - d_5 C_1}{C_1} \\ p_z - d_1 + d_6 \end{bmatrix} \tag{33}$$

$$(\text{左辺}) = \sqrt{p'^2_x + p'^2_z} \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} \tag{34}$$

$$\tag{35}$$

ただし，α は，

$$\alpha \coloneqq \mathrm{atan2}(p'_z, p'_x) \tag{36}$$

式 (32) の右辺を極座標形式で表現する．

$$(\text{右辺}) = \sqrt{l_2^2 + l_3^2 + 2 l_2 l_3 C_3} \begin{bmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} C_2 \\ S_2 \end{bmatrix} \tag{37}$$

$$= \sqrt{l_2^2 + l_3^2 + 2 l_2 l_3 C_3} \begin{bmatrix} \cos(\theta_2 + \beta) \\ \sin(\theta_2 + \beta) \end{bmatrix} \tag{38}$$

ただし，β は

$$\beta \coloneqq \mathrm{atan2}(l_3 S_3, l_3 C_3 + l_2) \tag{39}$$

以上より，

$$p_x'^2 + p_z'^2 = l_2^2 + l_3^2 + 2l_2l_3C_3 \tag{40}$$

$$\therefore C_3 = \frac{p_x'^2 + p_z'^2 - l_2^2 + l_3^2}{2l_2l_3C_3} \tag{41}$$

$$\therefore \theta_3 = \pm\mathrm{atan2}(\sqrt{1 - C_3^2}, C_3) \tag{42}$$

$$\theta_2 = \alpha - \beta \tag{43}$$

$$= \mathrm{atan2}(p_z', p_x') - \mathrm{atan2}(l_3S_3, l_3C_3 + l_2) \tag{44}$$

3.12 逆運動学により入力した位置に手先を移動させるプログラムを作成

逆運動学により入力した位置に手先を移動させるプログラムを作成し，シミュレータ上で動作を確認した．作成したプログラムは付録の Listing 5 に掲載しておく．

式 (42) の θ_3 の計算において，atan2 の符号は $-$ とした．また，atan2 と sqrt の実行前に，これらの関数が計算可能化を判定し，計算不可の場合はそれぞれエラーが発生するようにした．

エラーについて，atan2 の場合は分母の絶対値が 0.001 以下になる場合，ZeroDivisionError を raise するようにした．sqrt の場合，根号の中身が負になった場合，SqrtError を raise するようにした．

次の 4 つの手先位置を与え，シミュレータの動作を確認した．(a),(b) は動作範囲内かつエラーも出ない適切な手先位置である．これら 2 つの手先位置を与えた場合のロボットアームのシミュレーション上の挙動を図 6 に示す．(c) は atan2 の分母のゼロ割り，(d) は sqrt の中身が負になることを意図して設定した．それぞれの場合に出力されたエラーを以下に示す．

(a) ${}^0p_t = \begin{bmatrix} 150 & -100 & 70 & 1 \end{bmatrix}^\mathsf{T}$

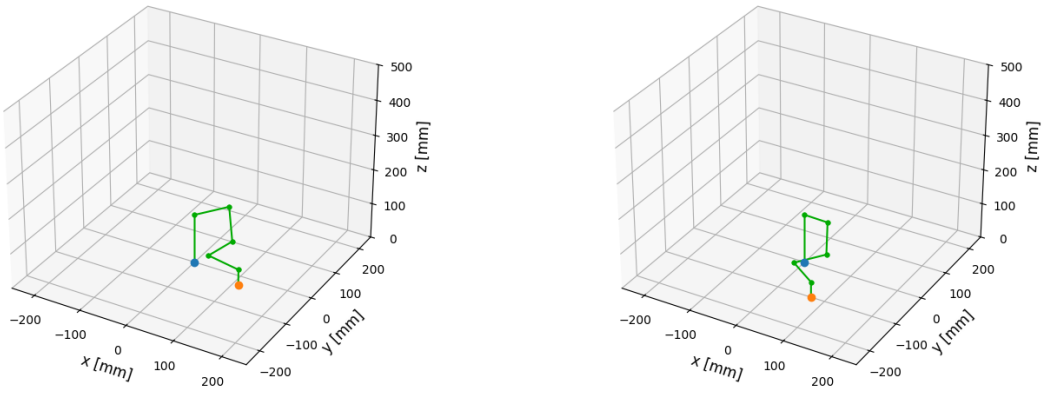
(b) ${}^0p_t = \begin{bmatrix} 100 & -150 & 50 & 1 \end{bmatrix}^\mathsf{T}$

(c) ${}^0p_t = \begin{bmatrix} 0 & 0 & 100 & 1 \end{bmatrix}^\mathsf{T}$

```
1         ZeroDivisionError: zero_division error
```

(d) ${}^0p_t = \begin{bmatrix} 1000 & 1000 & 100 & 1 \end{bmatrix}^\mathsf{T}$

```
1         SqrtError: sqrt error
```



(a) **(b)**

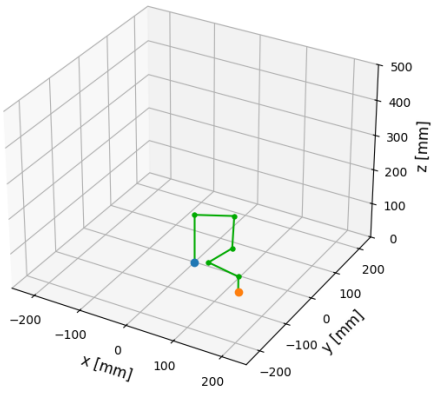
図 6: 適切な手先位置が与えられたときのロボットアームの挙動

3.13　初期位置にあるプレートを最終位置に移動させるプログラム

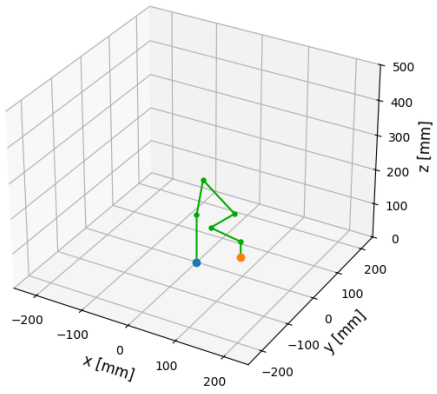
初期位置にあるプレートを最終位置に移動させるプログラムを作成した．作成したプログラムは付録の Listing 6 に掲載しておく．

今回は初期位置を ${}^0p_t = \begin{bmatrix} 150 & -100 & 50 & 1 \end{bmatrix}^\mathsf{T}$，最終位置を ${}^0p_t = \begin{bmatrix} 100 & -150 & 50 & 1 \end{bmatrix}^\mathsf{T}$ とした．運搬のときは，z 軸方向に 100 mm だけ持ち上げてから運搬するようにした．そのため，運搬時の軌跡は初期位置と最終位置，それからそれらの上空 100 mm の位置を結んだコの字型になる．基本的には 3.12 節を 4 回繰り返すようにして実装した．

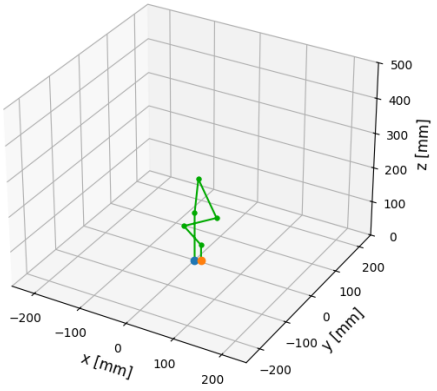
シミュレーション上での初期位置から最終位置までの運搬のようすを図 7 に示す．



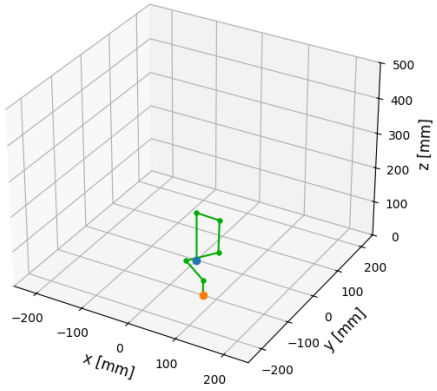
(a) 初期位置 $\begin{bmatrix}150 & -100 & 50 & 1\end{bmatrix}^{\mathsf{T}}$



(b) 初期位置上空 $\begin{bmatrix}150 & -100 & 150 & 1\end{bmatrix}^{\mathsf{T}}$



(c) 最終位置上空 $\begin{bmatrix}100 & -150 & 150 & 1\end{bmatrix}^{\mathsf{T}}$



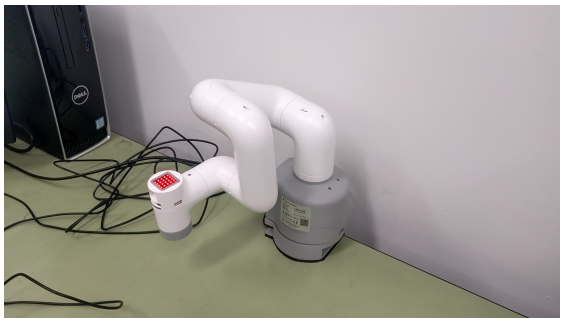
(d) 最終位置 $\begin{bmatrix}100 & -150 & 50 & 1\end{bmatrix}^{\mathsf{T}}$

図 7: 初期位置から最終位置までの運搬のようす（シミュレーション）

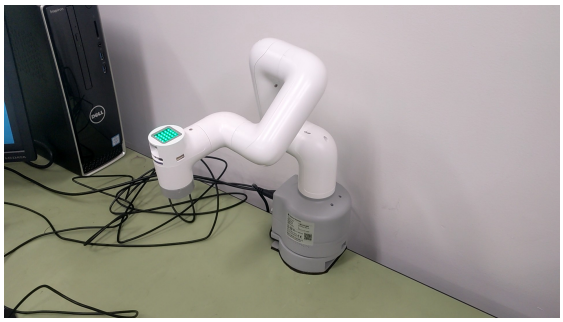
3.14　3.13 節を実機にて動作確認

3.13 節において作成したプログラムを実機で実行し，動作を確認した．また，運搬中の状態が視覚的にわかりやすいように，ロボットアーム付属の LED を初期位置から順に赤，緑，青，白で点灯するようにした．作成したプログラムは付録の Listing 7 に掲載しておく．

動作のようすを図 8 に示す．



(a) 初期位置 $\begin{bmatrix}150 & -100 & 50 & 1\end{bmatrix}^{\mathsf{T}}$



(b) 初期位置上空 $\begin{bmatrix}150 & -100 & 150 & 1\end{bmatrix}^{\mathsf{T}}$

(c) 最終位置上空 $\begin{bmatrix}100 & -150 & 150 & 1\end{bmatrix}^{\mathsf{T}}$

(d) 最終位置 $\begin{bmatrix}100 & -150 & 50 & 1\end{bmatrix}^{\mathsf{T}}$

図 8: 初期位置から最終位置までの運搬のようす（実機）

付録

Listing 1: 同次変換行列の積を計算するプログラム

```
import sympy

sympy.init_printing()
# sympy.var('C1, S1, C2, S2, C3, S3, C4, S4, C5, S5, C6, S6, d1, d4, d5, d6, l2, l3')
sympy.var("C1:7")
sympy.var("S1:7")
sympy.var("d1, d4, d5, d6")
sympy.var("l2, l3")

T1 = sympy.Matrix([
    [C1, 0, S1, 0],
    [S1, 0, -C1, 0],
    [0, 1, 0, d1],
    [0, 0, 0, 1]
])
T2 = sympy.Matrix([
    [C2, -S2, 0, l2*C2],
    [S2, C2, 0, l2*S2],
    [0, 0, 1, 0],
    [0, 0, 0, 1]
])
T3 = sympy.Matrix([
    [C3, -S3, 0, l3*C3],
    [S3, C3, 0, l3*S3],
    [0, 0, 1, 0],
    [0, 0, 0, 1]
])
T4 = sympy.Matrix([
    [C4, 0, S4, 0],
```

```
    [S4, 0, -C4, 0],
    [0, 1, 0, d4],
    [0, 0, 0, 1]
])
T5 = sympy.Matrix([
    [C5, 0, S5, 0],
    [S5, 0, -C5, 0],
    [0, 1, 0, d5],
    [0, 0, 0, 1]
])
T6 = sympy.Matrix([
    [C6, -S6, 0, 0],
    [S6, C6, 0, 0],
    [0, 0, 1, d6],
    [0, 0, 0, 1]
])

p6t = sympy.Matrix([0, 0, 0, 1])
p0t = T1*T2*T3*T4*T5*T6*p6t
# print(sympy.pretty(p0t))
print(sympy.pretty(sympy.simplify(p0t)))
```

Listing 2: 6 関節角度を入力し，全関節を同時に動作させるプログラム

```
import time
from math import radians,degrees,sin,cos,atan2,sqrt,pi,acos
import traceback

print("mode select:")
print("* 0 -> value check")
print("* 1 -> simulation")
print("* 2 -> move mode")
move_mode = int(input())


# ----- メイン関数 ----- #
def main():

    print("start program")

    try:    # try内で何らかのエラーが発生 -> 処理中断してexceptに移動

        # --- メインループ （実験内容に応じてここを変更） --- #
        while True:

            J = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  # 角度値の初期化（単位：degree）

            for i in range(6):
                print(f"input J[{i}]:")
                J[i] = float(input())               # 角度値をキーボード入力

            for i in range(6):                  # 6つの角度値を表示
                print("J"+str(i+1)+": ",J[i])

            moveto(J=J, marker_pos = [100, 100, 100])

    except:
        traceback.print_exc()                   # try内で発生したエラーを表示
# -------------------- #


# ----- 学生定義のサブ関数（実験内容に応じてここに関数を追加する） ----- #

# ------------------------------------------------------------ #

# ----- 【! 変更しないこと! 】mycobotライブラリの初期化 ----- #

if move_mode==1:
    from mycobot_sim import send_angles_sim

elif move_mode==2:
    print("load mycobot library...", end=" ")
    from pymycobot.mycobot import MyCobot
    from pymycobot.genre import Angle
    from pymycobot import PI_PORT, PI_BAUD

    mycobot = MyCobot(PI_PORT, PI_BAUD)
```

```
54      time.sleep(1)
55      mycobot.set_gripper_ini()
56      time.sleep(1)
57      print("OK")
58
59
60  # - 【! 変更しないこと! 】リンク長の定義 [mm] - #
61  d1 = 140
62  a2 = 110.4
63  a3 = 96.0
64  d4 = 66.39
65  d5 = 73.18
66  d6 = 43.6
67
68
69  # ----- 【! 変更しないこと! 】mycobot6軸関節確度制御----- #
70  def moveto(J, marker_pos):
71
72      angle_check(J)  # 角度が動作範囲内かチェック
73
74      if move_mode == 2:
75          print("move")
76          mycobot.send_angles([J[0]-90, J[1], J[2], J[3], J[4], J[5]], 20)
77          time.sleep(5)
78
79      elif move_mode == 1:
80          send_angles_sim(J=J, marker_pos=marker_pos)
81
82
83  # ----- 【! 変更しないこと! 】角度リミットエラー用 ----- #
84  class AngleError(Exception):
85      pass
86
87
88  # ----- 【! 変更しないこと! 】関節角度範囲チェック ----- #
89  def angle_check(J):
90
91      print("angle_check...", end=" ")
92
93      if J[0] < -90 or J[0] > 90:
94          raise AngleError('J1 angle error')
95
96      if J[1] < -120 or J[1] > 120:
97          raise AngleError('J2 angle error')
98
99      if J[2] < -150 or J[2] > 150:
100         raise AngleError('J3 angle error')
101
102     if J[3] < -120 or J[3] > 120:
103         raise AngleError('J4 angle error')
104
105     if J[4] < -120 or J[4] > 120:
106         raise AngleError('J5 angle error')
107
108     if J[5] < -90 or J[5] > 90:
109         raise AngleError('J6 angle error')
110
111     print("OK\n")
112
113 # ----- 【! 変更しないこと! 】メイン処理 ----- #
114 if __name__ == "__main__":
115     main()
```

Listing 3: 順運動学により，関節角度から手先位置を計算するプログラムの作成

```
1   import time
2   from math import radians,degrees,sin,cos,atan2,sqrt,pi,acos
3   import traceback
4   import numpy as np
5
6   print("mode select:")
7   print("* 0 -> value check")
8   print("* 1 -> simulation")
9   print("* 2 -> move mode")
10  move_mode = int(input())
11
12
```

```
# ----- メイン関数 ----- #
def main():

    print("start program")

    try:    # try内で何らかのエラーが発生 -> 処理中断してexceptに移動

        # --- メインループ （実験内容に応じてここを変更） --- #
        while True:

            J = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  # 角度値の初期化（単位：degree）

            # J = [-12.08, -51.17, -123.8, 84.96, 0.0, -12.08]
            # J = [-34.7, -63.36, -119.18, 92.54, 0.0, -34.7]
            # J = [26.27, -26.46, -146.28, 82.74, 0.0, 26.27]
            # J = [55.30, -51.17, -123.80, 84.96, 0.0, 55.30]
            # J = [77.92, -63.36, -119.18, 92.54, 0.0, 77.92]

            for i in range(6):
                print(f"input J[{i}]:")
                J[i] = float(input())               # 角度値をキーボード入力

            for i in range(6):                  # 6つの角度値を表示
                print("J"+str(i+1)+": ",J[i])

            # moveto(J=J, marker_pos=[100, 100, 100])
            moveto(J=J, marker_pos=calc_minions_pos(J))

    except:
        traceback.print_exc()                   # try内で発生したエラーを表示
# -------------------- #


# ----- 学生定義のサブ関数（実験内容に応じてここに関数を追加する） ----- #
def calc_minions_pos(J):
    # thetaのオフセット
    theta = np.array(J)/180*np.pi + [0, np.pi/2, 0, np.pi/2, -np.pi/2, 0]

    T1 = np.array([
        [np.cos(theta[0]), 0, np.sin(theta[0]), 0],
        [np.sin(theta[0]), 0, -np.cos(theta[0]), 0],
        [0, 1, 0, d1],
        [0, 0, 0, 1]
    ])
    T2 = np.array([
        [np.cos(theta[1]), -np.sin(theta[1]), 0, a2*np.cos(theta[1])],
        [np.sin(theta[1]), np.cos(theta[1]), 0, a2*np.sin(theta[1])],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ])
    T3 = np.array([
        [np.cos(theta[2]), -np.sin(theta[2]), 0, a3*np.cos(theta[2])],
        [np.sin(theta[2]), np.cos(theta[2]), 0, a3*np.sin(theta[2])],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ])
    T4 = np.array([
        [np.cos(theta[3]), 0, np.sin(theta[3]), 0],
        [np.sin(theta[3]), 0, -np.cos(theta[3]), 0],
        [0, 1, 0, d4],
        [0, 0, 0, 1]
    ])
    T5 = np.array([
        [np.cos(theta[4]), 0, np.sin(theta[4]), 0],
        [np.sin(theta[4]), 0, -np.cos(theta[4]), 0],
        [0, 1, 0, d5],
        [0, 0, 0, 1]
    ])
    T6 = np.array([
        [np.cos(theta[5]), -np.sin(theta[5]), 0, 0],
        [np.sin(theta[5]), np.cos(theta[5]), 0, 0],
        [0, 0, 1, d6],
        [0, 0, 0, 1]
    ])

    pos = T1@T2@T3@T4@T5@T6@np.array([0, 0, 0, 1])
    print(pos)
    return pos[:3].tolist()

```

```
92  # ------------------------------------------------------------ #
93
94  # ----- 【! 変更しないこと! 】mycobotライブラリの初期化 ----- #
95
96  if move_mode==1:
97      from mycobot_sim import send_angles_sim
98
99  elif move_mode==2:
100     print("load mycobot library...", end=" ")
101     from pymycobot.mycobot import MyCobot
102     from pymycobot.genre import Angle
103     from pymycobot import PI_PORT, PI_BAUD
104
105     mycobot = MyCobot(PI_PORT, PI_BAUD)
106     time.sleep(1)
107     mycobot.set_gripper_ini()
108     time.sleep(1)
109     print("OK")
110
111
112 # - 【! 変更しないこと! 】リンク長の定義 [mm] - #
113 d1 = 140
114 a2 = 110.4
115 a3 = 96.0
116 d4 = 66.39
117 d5 = 73.18
118 d6 = 43.6
119
120
121 # ----- 【! 変更しないこと! 】mycobot6軸関節確度制御----- #
122 def moveto(J, marker_pos):
123
124     angle_check(J)  # 角度が動作範囲内かチェック
125
126     if move_mode == 2:
127         print("move")
128         mycobot.send_angles([J[0]-90, J[1], J[2], J[3], J[4], J[5]], 20)
129         time.sleep(5)
130
131     elif move_mode == 1:
132         send_angles_sim(J=J, marker_pos=marker_pos)
133
134
135 # ----- 【! 変更しないこと! 】角度リミットエラー用 ----- #
136 class AngleError(Exception):
137     pass
138
139
140 # ----- 【! 変更しないこと! 】関節角度範囲チェック ----- #
141 def angle_check(J):
142
143     print("angle_check...", end=" ")
144
145     if J[0] < -90 or J[0] > 90:
146         raise AngleError('J1 angle error')
147
148     if J[1] < -120 or J[1] > 120:
149         raise AngleError('J2 angle error')
150
151     if J[2] < -150 or J[2] > 150:
152         raise AngleError('J3 angle error')
153
154     if J[3] < -120 or J[3] > 120:
155         raise AngleError('J4 angle error')
156
157     if J[4] < -120 or J[4] > 120:
158         raise AngleError('J5 angle error')
159
160     if J[5] < -90 or J[5] > 90:
161         raise AngleError('J6 angle error')
162
163     print("OK\n")
164
165 # ----- 【! 変更しないこと! 】メイン処理 ----- #
166 if __name__ == "__main__":
167     main()
```

Listing 4: Z 方向の手先位置が 15 mm 以内の場合にエラーを発生させるプログラム

```
import time
from math import radians,degrees,sin,cos,atan2,sqrt,pi,acos
import traceback
import numpy as np

print("mode select:")
print("* 0 -> value check")
print("* 1 -> simulation")
print("* 2 -> move mode")
move_mode = int(input())


# ----- メイン関数 ----- #
def main():

    print("start program")

    try:    # try内で何らかのエラーが発生 -> 処理中断してexceptに移動

        # --- メインループ （実験内容に応じてここを変更） --- #
        while True:

            J = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]  # 角度値の初期化（単位：degree）

            # J = [63.24, -41.72, -103.45, 55.17, 0.0, 63.24]
            # J = [63.24, -85.92, -83.17, 79.09, 0.0, 63.24]
            # J = [145.3, -63.36, -119.18, 92.54, 0.0, 145.3]
            # J = [41.6, -118.04, -155.24, 183.28, 0.0, 41.6]

            for i in range(6):
                print(f"input J[{i}]:")
                J[i] = float(input())               # 角度値をキーボード入力

            for i in range(6):                  # 6つの角度値を表示
                print("J"+str(i+1)+": ",J[i])

            # moveto(J=J, marker_pos=[100, 100, 100])
            moveto(J=J, marker_pos=calc_minions_pos(J))

    except:
        traceback.print_exc()                   # try内で発生したエラーを表示
# -------------------- #


# ----- 学生定義のサブ関数（実験内容に応じてここに関数を追加する） ----- #
def calc_minions_pos(J):
    # thetaのオフセット
    theta = np.array(J)/180*np.pi + [0, np.pi/2, 0, np.pi/2, -np.pi/2, 0]

    T1 = np.array([
        [np.cos(theta[0]), 0, np.sin(theta[0]), 0],
        [np.sin(theta[0]), 0, -np.cos(theta[0]), 0],
        [0, 1, 0, d1],
        [0, 0, 0, 1]
    ])
    T2 = np.array([
        [np.cos(theta[1]), -np.sin(theta[1]), 0, a2*np.cos(theta[1])],
        [np.sin(theta[1]), np.cos(theta[1]), 0, a2*np.sin(theta[1])],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ])
    T3 = np.array([
        [np.cos(theta[2]), -np.sin(theta[2]), 0, a3*np.cos(theta[2])],
        [np.sin(theta[2]), np.cos(theta[2]), 0, a3*np.sin(theta[2])],
        [0, 0, 1, 0],
        [0, 0, 0, 1]
    ])
    T4 = np.array([
        [np.cos(theta[3]), 0, np.sin(theta[3]), 0],
        [np.sin(theta[3]), 0, -np.cos(theta[3]), 0],
        [0, 1, 0, d4],
        [0, 0, 0, 1]
    ])
    T5 = np.array([
        [np.cos(theta[4]), 0, np.sin(theta[4]), 0],
        [np.sin(theta[4]), 0, -np.cos(theta[4]), 0],
```

```
77          [0, 1, 0, d5],
78          [0, 0, 0, 1]
79      ])
80      T6 = np.array([
81          [np.cos(theta[5]), -np.sin(theta[5]), 0, 0],
82          [np.sin(theta[5]), np.cos(theta[5]), 0, 0],
83          [0, 0, 1, d6],
84          [0, 0, 0, 1]
85      ])
86
87      pos = T1@T2@T3@T4@T5@T6@np.array([0, 0, 0, 1])
88      print(pos)
89      return pos[:3].tolist()
90
91  def z_check(pos):
92      print("z_check...", end=" ")
93
94      if pos[2] < 15.0:
95          raise ZError("z error")
96
97      print("OK\n")
98
99  class ZError(Exception):
100     pass
101
102 # ------------------------------------------------------------ #
103
104 # ----- 【! 変更しないこと! 】mycobotライブラリの初期化 ----- #
105
106 if move_mode==1:
107     from mycobot_sim import send_angles_sim
108
109 elif move_mode==2:
110     print("load mycobot library...", end=" ")
111     from pymycobot.mycobot import MyCobot
112     from pymycobot.genre import Angle
113     from pymycobot import PI_PORT, PI_BAUD
114
115     mycobot = MyCobot(PI_PORT, PI_BAUD)
116     time.sleep(1)
117     mycobot.set_gripper_ini()
118     time.sleep(1)
119     print("OK")
120
121
122 # - 【! 変更しないこと! 】リンク長の定義 [mm] - #
123 d1 = 140
124 a2 = 110.4
125 a3 = 96.0
126 d4 = 66.39
127 d5 = 73.18
128 d6 = 43.6
129
130
131 # ----- 【! 変更しないこと! 】mycobot6軸関節確度制御----- #
132 def moveto(J, marker_pos):
133
134     angle_check(J)  # 角度が動作範囲内かチェック
135     z_check(pos=marker_pos) # z手先位置が小さすぎないことをチェック
136
137     if move_mode == 2:
138         print("move")
139         mycobot.send_angles([J[0]-90, J[1], J[2], J[3], J[4], J[5]], 20)
140         time.sleep(5)
141
142     elif move_mode == 1:
143         send_angles_sim(J=J, marker_pos=marker_pos)
144
145
146 # ----- 【! 変更しないこと! 】角度リミットエラー用 ----- #
147 class AngleError(Exception):
148     pass
149
150
151 # ----- 【! 変更しないこと! 】関節角度範囲チェック ----- #
152 def angle_check(J):
153
154     print("angle_check...", end=" ")
155
```

```
156     if J[0] < -90 or J[0] > 90:
157         raise AngleError('J1 angle error')
158
159     if J[1] < -120 or J[1] > 120:
160         raise AngleError('J2 angle error')
161
162     if J[2] < -150 or J[2] > 150:
163         raise AngleError('J3 angle error')
164
165     if J[3] < -120 or J[3] > 120:
166         raise AngleError('J4 angle error')
167
168     if J[4] < -120 or J[4] > 120:
169         raise AngleError('J5 angle error')
170
171     if J[5] < -90 or J[5] > 90:
172         raise AngleError('J6 angle error')
173
174     print("OK\n")
175
176 # ----- 【! 変更しないこと! 】メイン処理 ----- #
177 if __name__ == "__main__":
178     main()
```

Listing 5: 逆運動学により入力した位置に手先を移動させるプログラム

```
1   import time
2   # from math import radians,degrees,sin,cos,atan2,sqrt,pi,acos
3   import traceback
4   import numpy as np
5
6   print("mode select:")
7   print("* 0 -> value check")
8   print("* 1 -> simulation")
9   print("* 2 -> move mode")
10  move_mode = int(input())
11
12
13  # ----- メイン関数 ----- #
14  def main():
15
16      print("start program")
17
18      try:    # try内で何らかのエラーが発生 -> 処理中断してexceptに移動
19
20          # --- メインループ （実験内容に応じてここを変更） --- #
21          while True:
22              # 座標を入力
23              print(f"input x:")
24              x = float(input())
25              print(f"input y:")
26              y = float(input())
27              print(f"input z:")
28              z = float(input())
29              print(f"x = {x}")
30              print(f"y = {y}")
31              print(f"z = {z}")
32
33
34              J = calc_inverse_kinematics(x, y, z)
35              for i in range(6):                  # 6つの角度値を表示
36                  print("J"+str(i+1)+": ",J[i])
37
38              moveto(J=J, marker_pos=[x, y, z])
39
40      except:
41          traceback.print_exc()                   # try内で発生したエラーを表示
42  # -------------------- #
43
44
45  # ----- 学生定義のサブ関数（実験内容に応じてここに関数を追加する） ----- #
46  def calc_inverse_kinematics(x, y, z):
47      d1 = 140
48      a2 = 110.4
49      a3 = 96.0
50      d4 = 66.39
51      d5 = 73.18
```

```
52      d6 = 43.6
53
54      theta = np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
55
56      # 逆運動学の計算
57      zero_division_check(x*x + y*y)
58      if y > 0:
59          theta[0] = np.pi - np.atan2(x, y) - np.acos(d4 / np.sqrt(x*x + y*y))
60      else:
61          theta[0] = -(np.pi/2 - np.atan2(x, -y) - np.asin(d4 / np.sqrt(x*x + y*y)))
62
63      X = (x - d5*np.cos(theta[0]) - d4*np.sin(theta[0])) / np.cos(theta[0])
64      Z = z + d6 - d1
65
66      sqrt_check(1 - ((X*X + Z*Z - a2*a2 - a3*a3)/(2*a2*a3))**2)
67      theta[2] = -np.atan2(np.sqrt(1 - ((X*X + Z*Z - a2*a2 - a3*a3)/(2*a2*a3))**2), (X*X + Z*Z - a2*a2 - a3*
          a3)/(2*a2*a3))
68
69      alpha = np.atan2(Z, X)
70      beta = np.atan2(a3*np.sin(theta[2]), a3*np.cos(theta[2]) + a2)
71      theta[1] = alpha - beta
72
73      theta[3] = np.pi/2 - theta[1] - theta[2]
74      theta[4] = -np.pi/2
75      theta[5] = theta[0]
76
77      J = 180/np.pi*(theta - np.array([0, np.pi/2, 0, np.pi/2, -np.pi/2, 0]))
78
79      return J.tolist()
80
81  def z_check(pos):
82      print("z_check...", end=" ")
83
84      if pos[2] < 15.0:
85          raise ZError("z error")
86
87      print("OK\n")
88  def zero_division_check(val):
89      print("zero_division_check...", end=" ")
90
91      if np.abs(val) < 0.001:
92          raise ZeroDivisionError("zero_division error")
93
94      print("OK\n")
95  def sqrt_check(val):
96      print("sqrt_check...", end=" ")
97
98      if val < 0:
99          raise SqrtError("sqrt error")
100
101     print("OK\n")
102
103 class ZError(Exception):
104     pass
105 class ZeroDivisionError(Exception):
106     pass
107 class SqrtError(Exception):
108     pass
109
110 # ------------------------------------------------------------ #
111
112 # ----- 【! 変更しないこと! 】mycobotライブラリの初期化 ----- #
113
114 if move_mode==1:
115     from mycobot_sim import send_angles_sim
116
117 elif move_mode==2:
118     print("load mycobot library...", end=" ")
119     from pymycobot.mycobot import MyCobot
120     from pymycobot.genre import Angle
121     from pymycobot import PI_PORT, PI_BAUD
122
123     mycobot = MyCobot(PI_PORT, PI_BAUD)
124     time.sleep(1)
125     mycobot.set_gripper_ini()
126     time.sleep(1)
127     print("OK")
128
129
```

```
# - 【! 変更しないこと! 】リンク長の定義 [mm] - #
d1 = 140
a2 = 110.4
a3 = 96.0
d4 = 66.39
d5 = 73.18
d6 = 43.6


# ----- 【! 変更しないこと! 】mycobot6軸関節確度制御----- #
def moveto(J, marker_pos):

    angle_check(J)  # 角度が動作範囲内かチェック
    z_check(pos=marker_pos) # z手先位置が小さすぎないことをチェック

    if move_mode == 2:
        print("move")
        mycobot.send_angles([J[0]-90, J[1], J[2], J[3], J[4], J[5]], 20)
        time.sleep(5)

    elif move_mode == 1:
        send_angles_sim(J=J, marker_pos=marker_pos)


# ----- 【! 変更しないこと! 】角度リミットエラー用 ----- #
class AngleError(Exception):
    pass


# ----- 【! 変更しないこと! 】関節角度範囲チェック ----- #
def angle_check(J):

    print("angle_check...", end=" ")

    if J[0] < -90 or J[0] > 90:
        raise AngleError('J1 angle error')

    if J[1] < -120 or J[1] > 120:
        raise AngleError('J2 angle error')

    if J[2] < -150 or J[2] > 150:
        raise AngleError('J3 angle error')

    if J[3] < -120 or J[3] > 120:
        raise AngleError('J4 angle error')

    if J[4] < -120 or J[4] > 120:
        raise AngleError('J5 angle error')

    if J[5] < -90 or J[5] > 90:
        raise AngleError('J6 angle error')

    print("OK\n")

# ----- 【! 変更しないこと! 】メイン処理 ----- #
if __name__ == "__main__":
    main()
```

Listing 6: 初期位置にあるプレートを最終位置に移動させるプログラム

```
import time
# from math import radians,degrees,sin,cos,atan2,sqrt,pi,acos
import traceback
import numpy as np

print("mode select:")
print("* 0 -> value check")
print("* 1 -> simulation")
print("* 2 -> move mode")
move_mode = int(input())


# ----- メイン関数 ----- #
def main():

    print("start program")

```

```
18      try:    # try内で何らかのエラーが発生 -> 処理中断してexceptに移動
19
20          # --- メインループ （実験内容に応じてここを変更） --- #
21          r0 = np.array([150, -100, 50])
22          r1 = np.array([100, -150, 50])
23
24          R = np.array([r0, r0 + np.array([0, 0, 100]), r1 + np.array([0, 0, 100]), r1]).T
25          print(R)
26
27          for i in range(4):
28              J = calc_inverse_kinematics(R[0][i], R[1][i], R[2][i])
29              for j in range(6):                  # 6つの角度値を表示
30                  # print("J"+str(j+1)+": ",J[j])
31                  print(f"J = {j+1}: {J[j]}")
32
33              moveto(J=J, marker_pos=[R[0][i], R[1][i], R[2][i]])
34
35      except:
36          traceback.print_exc()                   # try内で発生したエラーを表示
37  # -------------------- #
38
39
40  # ----- 学生定義のサブ関数（実験内容に応じてここに関数を追加する） ----- #
41  def calc_inverse_kinematics(x, y, z):
42      d1 = 140
43      a2 = 110.4
44      a3 = 96.0
45      d4 = 66.39
46      d5 = 73.18
47      d6 = 43.6
48
49      theta = np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
50
51      # 逆運動学の計算
52      zero_division_check(x*x + y*y)
53      if y > 0:
54          theta[0] = np.pi - np.arctan2(x, y) - np.arccos(d4 / np.sqrt(x*x + y*y))
55      else:
56          theta[0] = -(np.pi/2 - np.arctan2(x, -y) - np.arcsin(d4 / np.sqrt(x*x + y*y)))
57
58      X = (x - d5*np.cos(theta[0]) - d4*np.sin(theta[0])) / np.cos(theta[0])
59      Z = z + d6 - d1
60
61      sqrt_check(1 - ((X*X + Z*Z - a2*a2 - a3*a3)/(2*a2*a3))**2)
62      theta[2] = -np.arctan2(np.sqrt(1 - ((X*X + Z*Z - a2*a2 - a3*a3)/(2*a2*a3))**2), (X*X + Z*Z - a2*a2 - a3
          *a3)/(2*a2*a3))
63
64      alpha = np.arctan2(Z, X)
65      beta = np.arctan2(a3*np.sin(theta[2]), a3*np.cos(theta[2]) + a2)
66      theta[1] = alpha - beta
67
68      theta[3] = np.pi/2 - theta[1] - theta[2]
69      theta[4] = -np.pi/2
70      theta[5] = theta[0]
71
72      J = 180/np.pi*(theta - np.array([0, np.pi/2, 0, np.pi/2, -np.pi/2, 0]))
73
74      return J.tolist()
75
76  def z_check(pos):
77      print("z_check...", end=" ")
78
79      if pos[2] < 15.0:
80          raise ZError("z error")
81
82      print("OK\n")
83  def zero_division_check(val):
84      print("zero_division_check...", end=" ")
85
86      if np.abs(val) < 0.001:
87          raise ZeroDivisionError("zero_division error")
88
89      print("OK\n")
90  def sqrt_check(val):
91      print("sqrt_check...", end=" ")
92
93      if val < 0:
94          raise SqrtError("sqrt error")
95
```

```
96      print("OK\n")
97
98  class ZError(Exception):
99      pass
100 class ZeroDivisionError(Exception):
101     pass
102 class SqrtError(Exception):
103     pass
104
105 # ------------------------------------------------------------ #
106
107 # ----- 【! 変更しないこと! 】mycobotライブラリの初期化 ----- #
108
109 if move_mode==1:
110     from mycobot_sim import send_angles_sim
111
112 elif move_mode==2:
113     print("load mycobot library...", end=" ")
114     from pymycobot.mycobot import MyCobot
115     from pymycobot.genre import Angle
116     from pymycobot import PI_PORT, PI_BAUD
117
118     mycobot = MyCobot(PI_PORT, PI_BAUD)
119     time.sleep(1)
120     mycobot.set_gripper_ini()
121     time.sleep(1)
122     print("OK")
123
124
125 # - 【! 変更しないこと! 】リンク長の定義 [mm] - #
126 d1 = 140
127 a2 = 110.4
128 a3 = 96.0
129 d4 = 66.39
130 d5 = 73.18
131 d6 = 43.6
132
133
134 # ----- 【! 変更しないこと! 】mycobot6軸関節確度制御----- #
135 def moveto(J, marker_pos):
136
137     angle_check(J)  # 角度が動作範囲内かチェック
138     z_check(pos=marker_pos) # z手先位置が小さすぎないことをチェック
139
140     if move_mode == 2:
141         print("move")
142         mycobot.send_angles([J[0]-90, J[1], J[2], J[3], J[4], J[5]], 20)
143         time.sleep(5)
144
145     elif move_mode == 1:
146         send_angles_sim(J=J, marker_pos=marker_pos)
147
148
149 # ----- 【! 変更しないこと! 】角度リミットエラー用 ----- #
150 class AngleError(Exception):
151     pass
152
153
154 # ----- 【! 変更しないこと! 】関節角度範囲チェック ----- #
155 def angle_check(J):
156
157     print("angle_check...", end=" ")
158
159     if J[0] < -90 or J[0] > 90:
160         raise AngleError('J1 angle error')
161
162     if J[1] < -120 or J[1] > 120:
163         raise AngleError('J2 angle error')
164
165     if J[2] < -150 or J[2] > 150:
166         raise AngleError('J3 angle error')
167
168     if J[3] < -120 or J[3] > 120:
169         raise AngleError('J4 angle error')
170
171     if J[4] < -120 or J[4] > 120:
172         raise AngleError('J5 angle error')
173
174     if J[5] < -90 or J[5] > 90:
```

```
175         raise AngleError('J6 angle error')
176
177     print("OK\n")
178
179 # ----- 【! 変更しないこと! 】メイン処理 ----- #
180 if __name__ == "__main__":
181     main()
```

Listing 7: 初期位置にあるプレートを最終位置に移動させるプログラム（実機）

```
1  import time
2  # from math import radians,degrees,sin,cos,atan2,sqrt,pi,acos
3  import traceback
4  import numpy as np
5
6  print("mode select:")
7  print("* 0 -> value check")
8  print("* 1 -> simulation")
9  print("* 2 -> move mode")
10 move_mode = int(input())
11
12
13 # ----- メイン関数 ----- #
14 def main():
15
16     print("start program")
17
18     try:    # try内で何らかのエラーが発生 -> 処理中断してexceptに移動
19
20         # --- メインループ （実験内容に応じてここを変更） --- #
21         r0 = np.array([150, -100, 50])
22         r1 = np.array([100, -150, 50])
23
24         R = np.array([r0, r0 + np.array([0, 0, 100]), r1 + np.array([0, 0, 100]), r1]).T
25         print(R)
26
27         color = np.array([
28             [255, 0, 0],
29             [0, 255, 0],
30             [0, 0, 255],
31             [255, 255, 255]
32         ])
33
34         for i in range(4):
35             J = calc_inverse_kinematics(R[0][i], R[1][i], R[2][i])
36             for j in range(6):                  # 6つの角度値を表示
37                 print(f"J = {j+1}: {J[j]}")
38
39             if move_mode == 2:
40                 mycobot.set_color(color[i][0], color[i][1], color[i][2])
41             moveto(J=J, marker_pos=[R[0][i], R[1][i], R[2][i]])
42
43     except:
44         traceback.print_exc()                   # try内で発生したエラーを表示
45 # -------------------- #
46
47
48 # ----- 学生定義のサブ関数（実験内容に応じてここに関数を追加する） ----- #
49 def calc_inverse_kinematics(x, y, z):
50     d1 = 140
51     a2 = 110.4
52     a3 = 96.0
53     d4 = 66.39
54     d5 = 73.18
55     d6 = 43.6
56
57     theta = np.array([0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
58
59     # 逆運動学の計算
60     zero_division_check(x*x + y*y)
61     if y > 0:
62         theta[0] = np.pi - np.arctan2(x, y) - np.arccos(d4 / np.sqrt(x*x + y*y))
63     else:
64         theta[0] = -(np.pi/2 - np.arctan2(x, -y) - np.arcsin(d4 / np.sqrt(x*x + y*y)))
65
66     X = (x - d5*np.cos(theta[0]) - d4*np.sin(theta[0])) / np.cos(theta[0])
67     Z = z + d6 - d1
```

```
68
69      sqrt_check(1 - ((X*X + Z*Z - a2*a2 - a3*a3)/(2*a2*a3))**2)
70      theta[2] = -np.arctan2(np.sqrt(1 - ((X*X + Z*Z - a2*a2 - a3*a3)/(2*a2*a3))**2), (X*X + Z*Z - a2*a2 - a3
          *a3)/(2*a2*a3))
71
72      alpha = np.arctan2(Z, X)
73      beta = np.arctan2(a3*np.sin(theta[2]), a3*np.cos(theta[2]) + a2)
74      theta[1] = alpha - beta
75
76      theta[3] = np.pi/2 - theta[1] - theta[2]
77      theta[4] = -np.pi/2
78      theta[5] = theta[0]
79
80      J = 180/np.pi*(theta - np.array([0, np.pi/2, 0, np.pi/2, -np.pi/2, 0]))
81
82      return J.tolist()
83
84  def z_check(pos):
85      print("z_check...", end=" ")
86
87      if pos[2] < 15.0:
88          raise ZError("z error")
89
90      print("OK\n")
91  def zero_division_check(val):
92      print("zero_division_check...", end=" ")
93
94      if np.abs(val) < 0.001:
95          raise ZError("zero_division error")
96
97      print("OK\n")
98  def sqrt_check(val):
99      print("sqrt_check...", end=" ")
100
101     if val < 0:
102         raise ZError("sqrt error")
103
104     print("OK\n")
105
106 class ZError(Exception):
107     pass
108 class ZeroDivisionError(Exception):
109     pass
110 class SqrtError(Exception):
111     pass
112
113 # ------------------------------------------------------------ #
114
115 # ----- 【! 変更しないこと! 】mycobotライブラリの初期化 ----- #
116
117 if move_mode==1:
118     from mycobot_sim import send_angles_sim
119
120 elif move_mode==2:
121     print("load mycobot library...", end=" ")
122     from pymycobot.mycobot import MyCobot
123     from pymycobot.genre import Angle
124     from pymycobot import PI_PORT, PI_BAUD
125
126     mycobot = MyCobot(PI_PORT, PI_BAUD)
127     time.sleep(1)
128     mycobot.set_gripper_ini()
129     time.sleep(1)
130     print("OK")
131
132
133 # - 【! 変更しないこと! 】リンク長の定義 [mm] - #
134 d1 = 140
135 a2 = 110.4
136 a3 = 96.0
137 d4 = 66.39
138 d5 = 73.18
139 d6 = 43.6
140
141
142 # ----- 【! 変更しないこと! 】mycobot6軸関節確度制御----- #
143 def moveto(J, marker_pos):
144
145     angle_check(J)  # 角度が動作範囲内かチェック
```

```
146     z_check(pos=marker_pos) # z手先位置が小さすぎないことをチェック
147
148     if move_mode == 2:
149         print("move")
150         mycobot.send_angles([J[0]-90, J[1], J[2], J[3], J[4], J[5]], 20)
151         time.sleep(5)
152
153     elif move_mode == 1:
154         send_angles_sim(J=J, marker_pos=marker_pos)
155
156
157 # ----- 【! 変更しないこと! 】角度リミットエラー用 ----- #
158 class AngleError(Exception):
159     pass
160
161
162 # ----- 【! 変更しないこと! 】関節角度範囲チェック ----- #
163 def angle_check(J):
164
165     print("angle_check...", end=" ")
166
167     if J[0] < -90 or J[0] > 90:
168         raise AngleError('J1 angle error')
169
170     if J[1] < -120 or J[1] > 120:
171         raise AngleError('J2 angle error')
172
173     if J[2] < -150 or J[2] > 150:
174         raise AngleError('J3 angle error')
175
176     if J[3] < -120 or J[3] > 120:
177         raise AngleError('J4 angle error')
178
179     if J[4] < -120 or J[4] > 120:
180         raise AngleError('J5 angle error')
181
182     if J[5] < -90 or J[5] > 90:
183         raise AngleError('J6 angle error')
184
185     print("OK\n")
186
187 # ----- 【! 変更しないこと! 】メイン処理 ----- #
188 if __name__ == "__main__":
189     main()
```