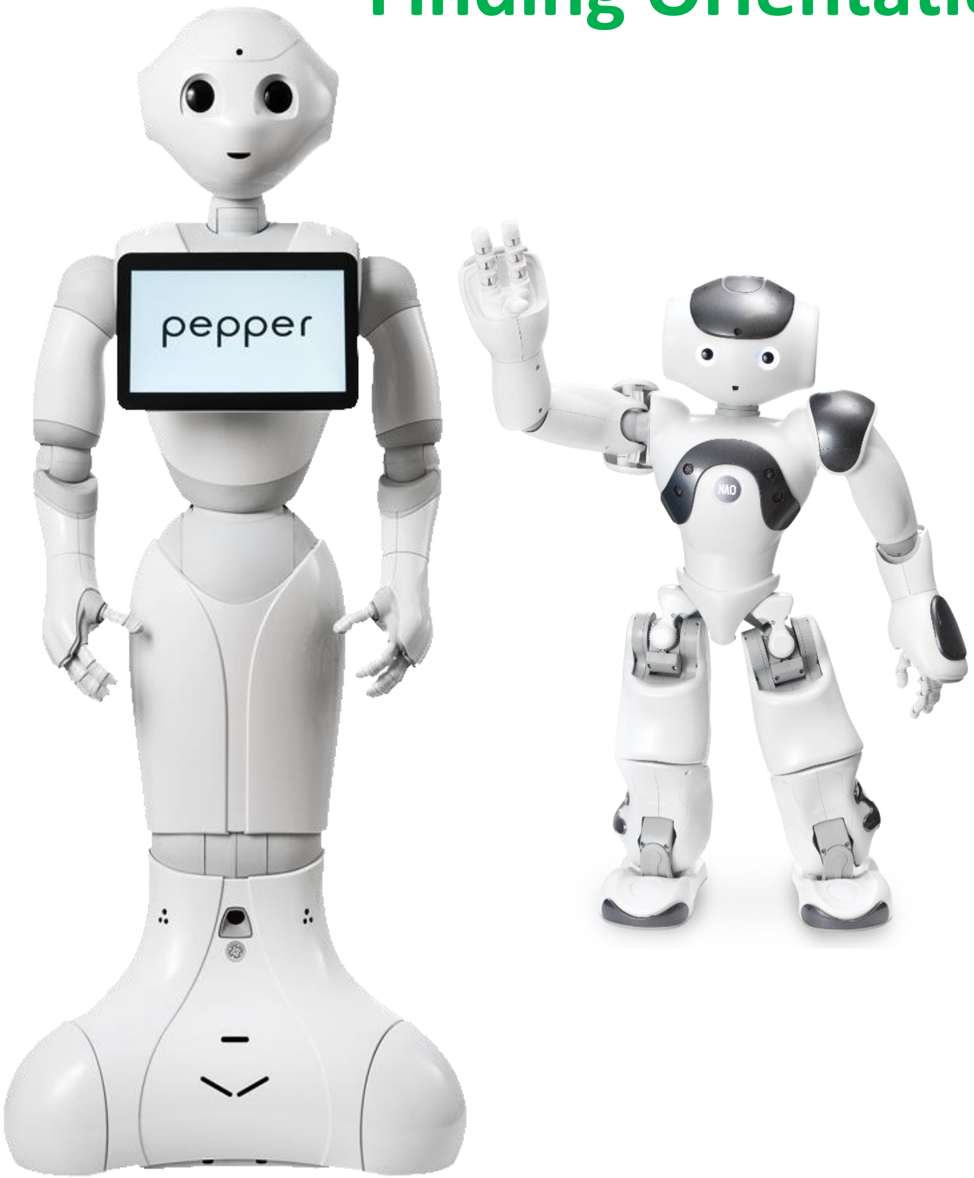
Applied Robotics

Robot Manipulators 09

Finding Orientation of a robot

Inverse Kinematics 6 DoF



Inverse Kinematics till now

- Previously, we were finding position of an end effector for 3 DoF manipulators using top and side views and trigonometry.
- If we try the same for inverse kinematics then it is going to be very complicated and will get more complicated with the addition of DoF.
- To simplify this, we use a modified method to find the position of manipulator with more than 3 DoF.

Assumption for method with more than 3 DoF

- In order to work with this method, we need to make an assumption:

"The first three joints are entirely responsible for positioning (in three linear axes) the end effector, and any additional joints are responsible for orienting (rotating) the end-effector".

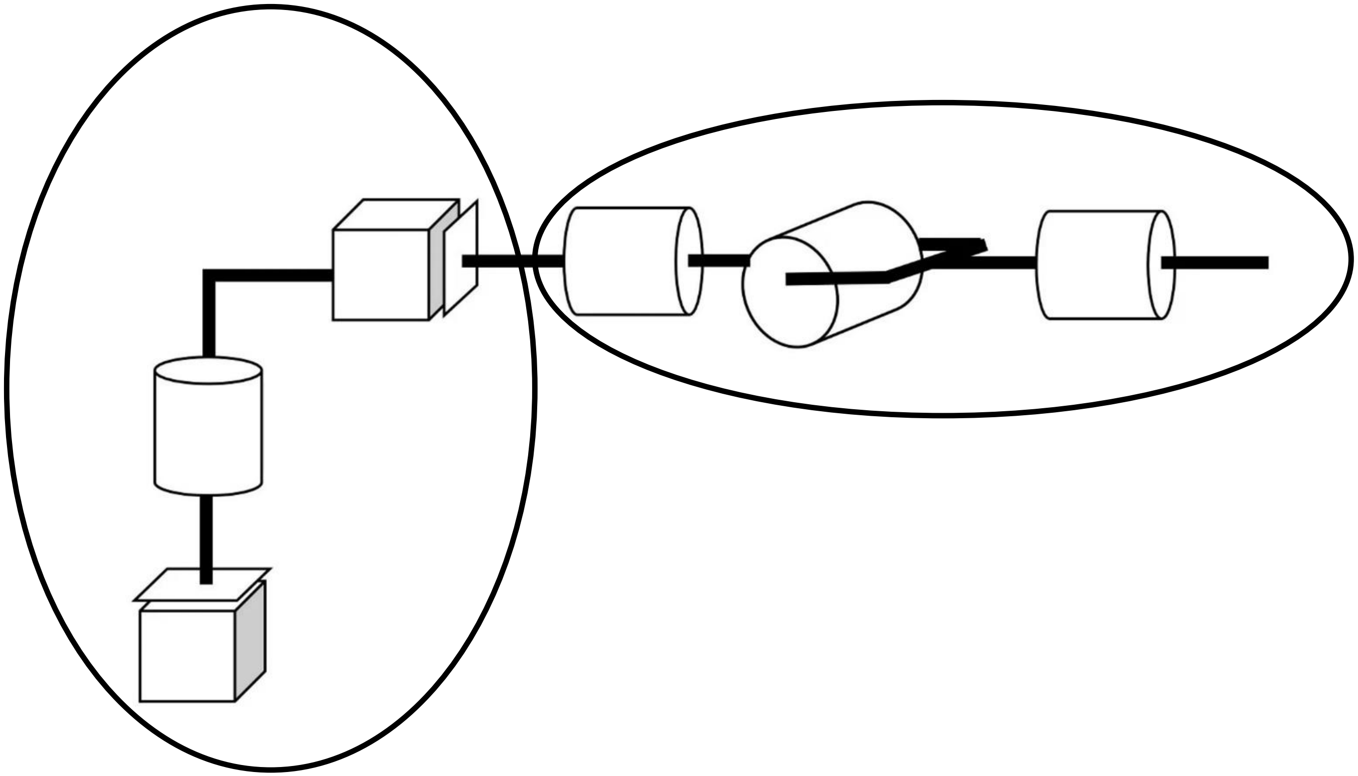
- Spherical wrist can provide you three rotations (pitch, yaw and roll)
- This is one of the reason a spherical wrist is most commonly used in the industry.
- The spherical wrist is design to have all links length as close to zero to have no effect on the linear positioning of the end effector.

6 DoF procedure break down

- We break down 6 DoF inverse kinematic procedure into 7 steps.
- **Step 01**: Draw a kinematic diagram of only for the first 3 joints, and do inverse kinematics for the position.
- **Step 02**: Do forward kinematics on the first three joints to get the rotation part only R0_3
- **Step 03**: Find the matrix inverse of the R 0_3 matrix
- **Step 04**: Do forward kinematics on the last three joints & pull out the rotation part R 3_6
- **Step 05**: Specify what you want the rotation matrix R 0_6 to be?
- **Step 06**: Given a derived x, y and z position, solve for the first three joints using the inverse kinematics equations from step 01
- **Step 07**: Plug in those variables (from Step 06) and use the rotation matrix to solve for the last three joints

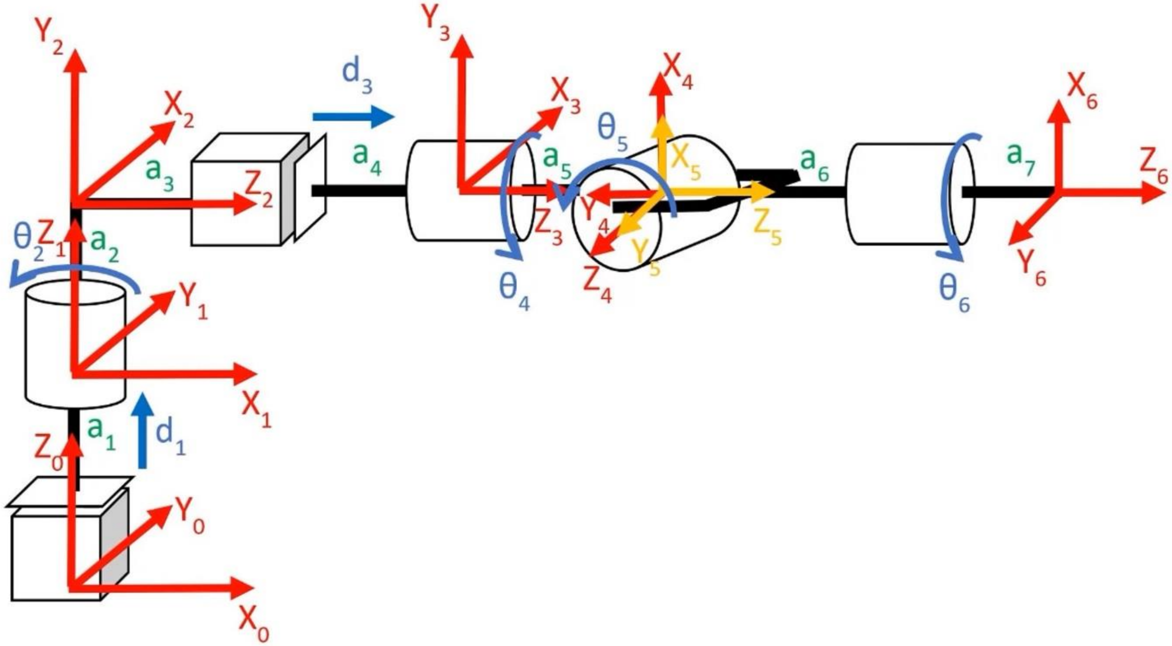
6 DoF procedure break down

- Below is the cylindrical manipulator with spherical wrist on the top of it.



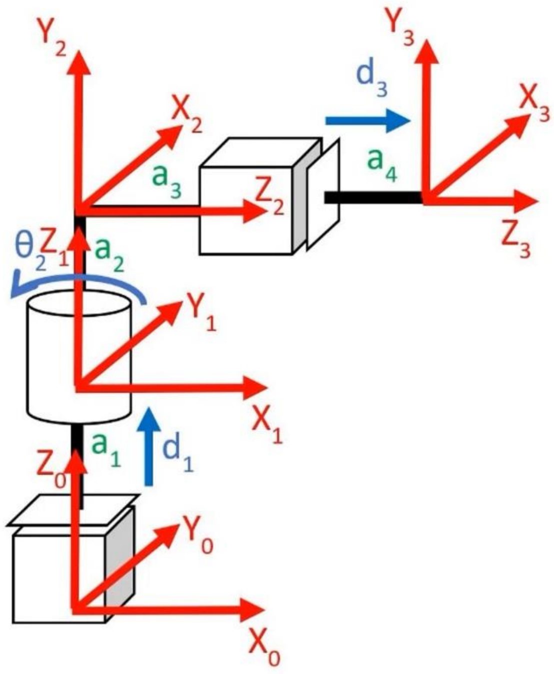
6 DoF procedure break down

- Lets draw frames over Kinematic diagram using Denavit-Hartenberg rules.



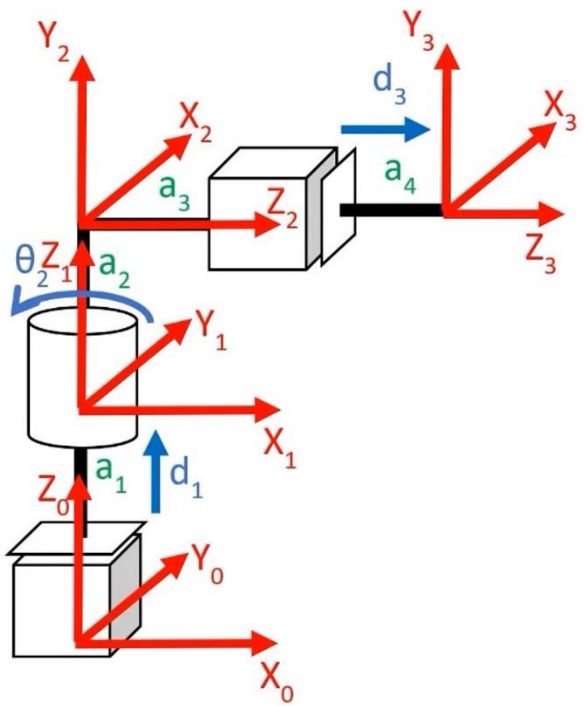
6 DoF procedure break down

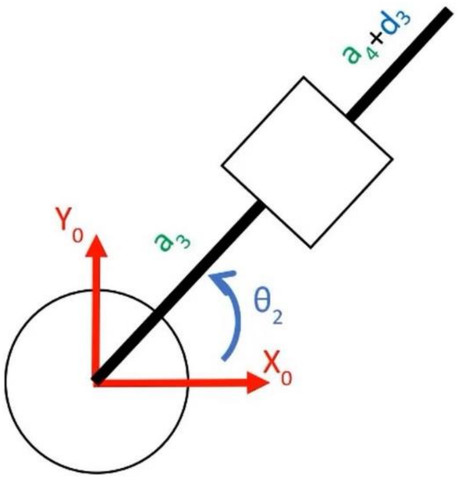
- **Step 01**: Draw a kinematic diagram of only for the first 3 joints, and do inverse kinematics for position.



6 DoF procedure break down

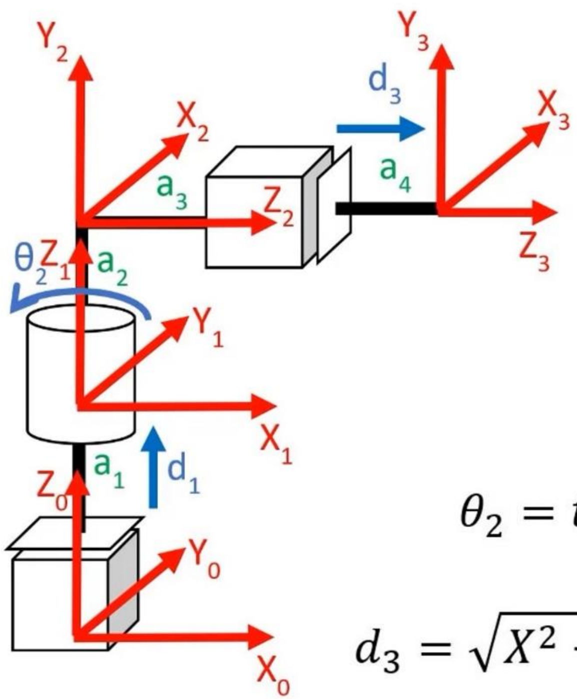
- **Step 01**: Draw a kinematic diagram of only for the first 3 joints, and do inverse kinematics for position.

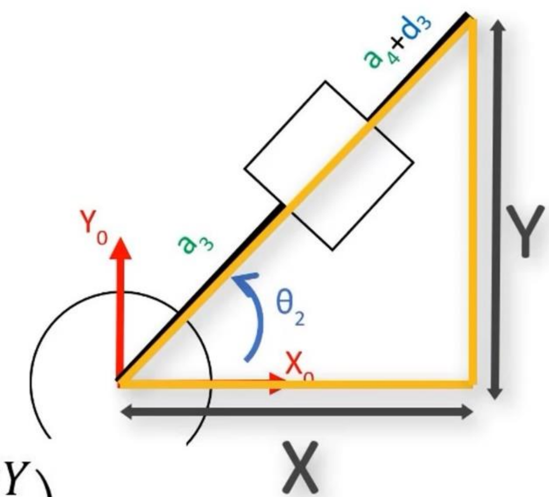




6 DoF procedure break down

- **Step 01**: Draw a kinematic diagram of only for the first 3 joints, and do inverse kinematics for position.



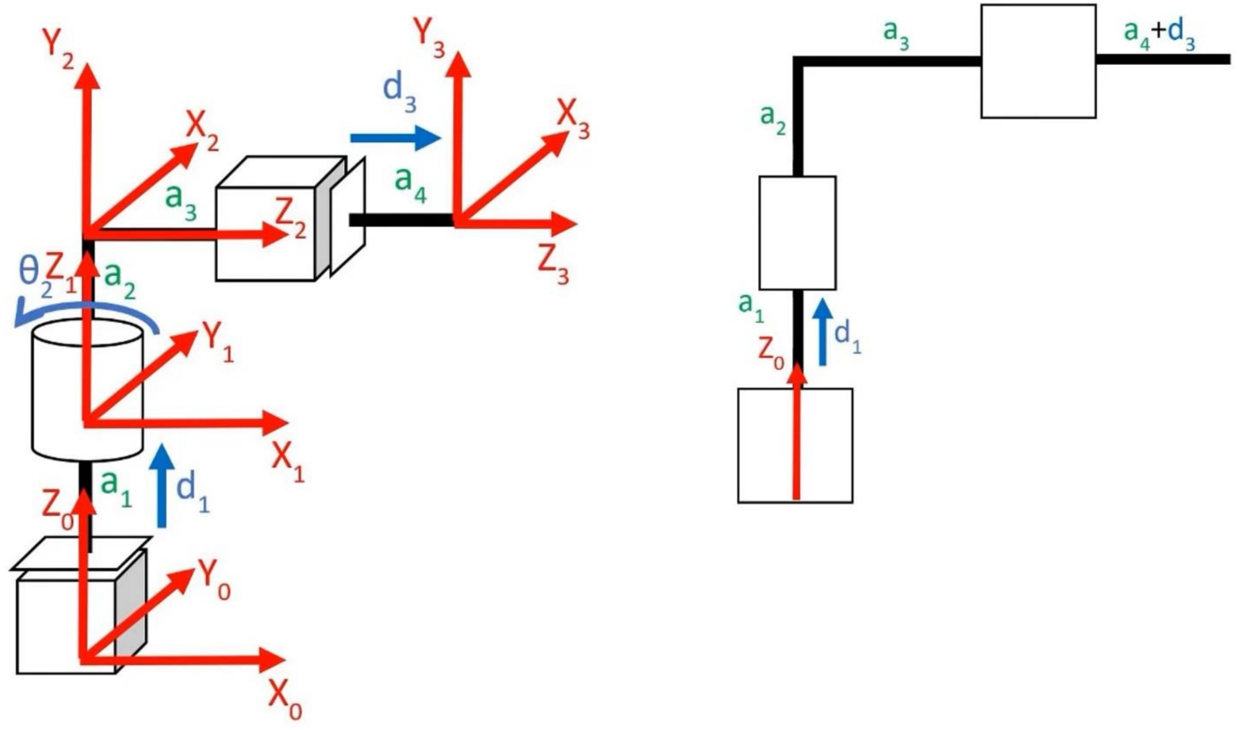


$$\theta_2 = tan^{-1}\left(\frac{Y}{X}\right)$$

$$d_3 = \sqrt{X^2 + Y^2} - a_3 - a_4$$

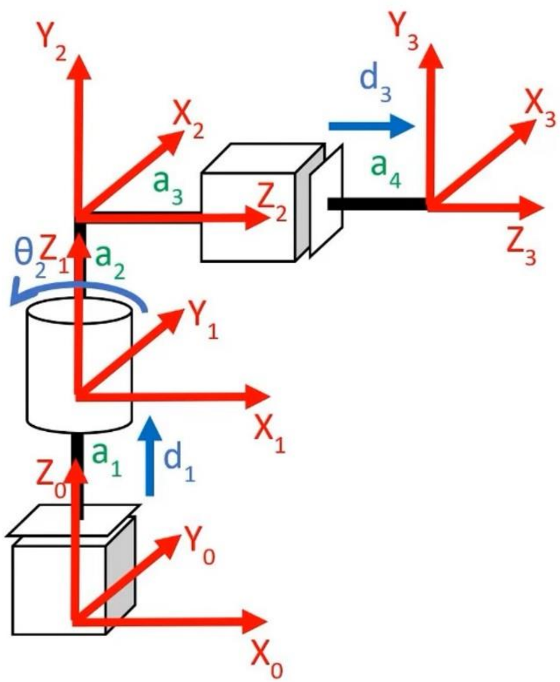
6 DoF procedure break down

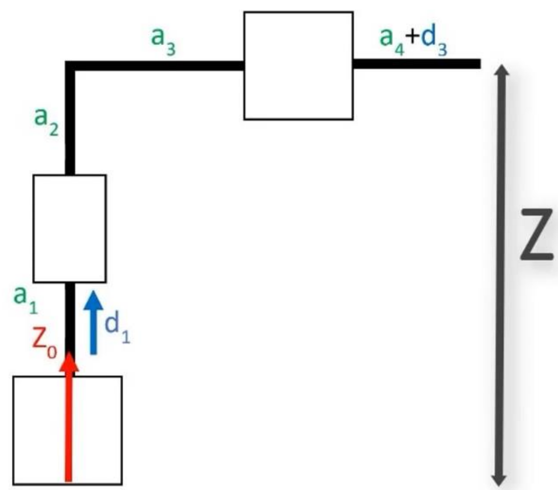
- **Step 01**: Draw a kinematic diagram of only for the first 3 joints, and do inverse kinematics for position.



6 DoF procedure break down

- **Step 01**: Draw a kinematic diagram of only for the first 3 joints, and do inverse kinematics for position.

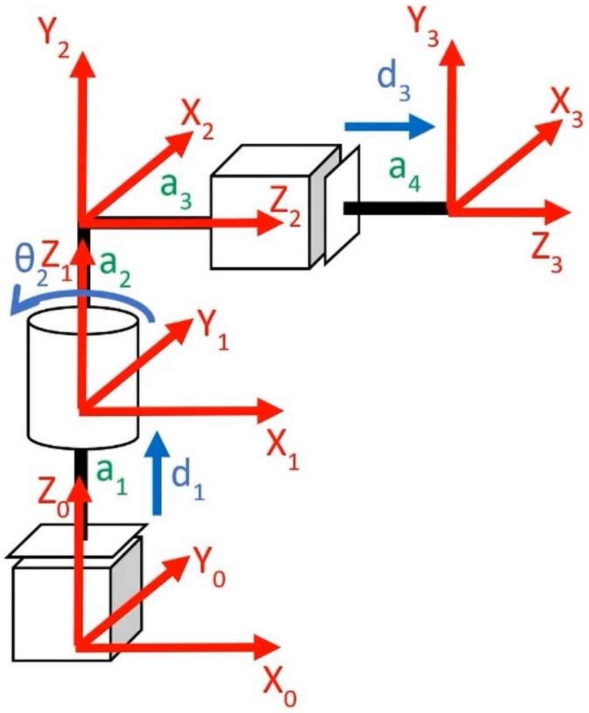




$$d_1 = Z - a_2 - a_1$$

6 DoF procedure break down

- **Step 02**: Do forward kinematics on the first three joints to get the rotation part only R0_3.



$$R_3^0 = \begin{bmatrix} -S\theta_2 & 0 & C\theta_2 \\ C\theta_2 & 0 & S\theta_2 \\ 0 & 1 & 0 \end{bmatrix}$$

6 DoF procedure break down

- **Step 03**: Find the inverse of the R 0_3 matrix

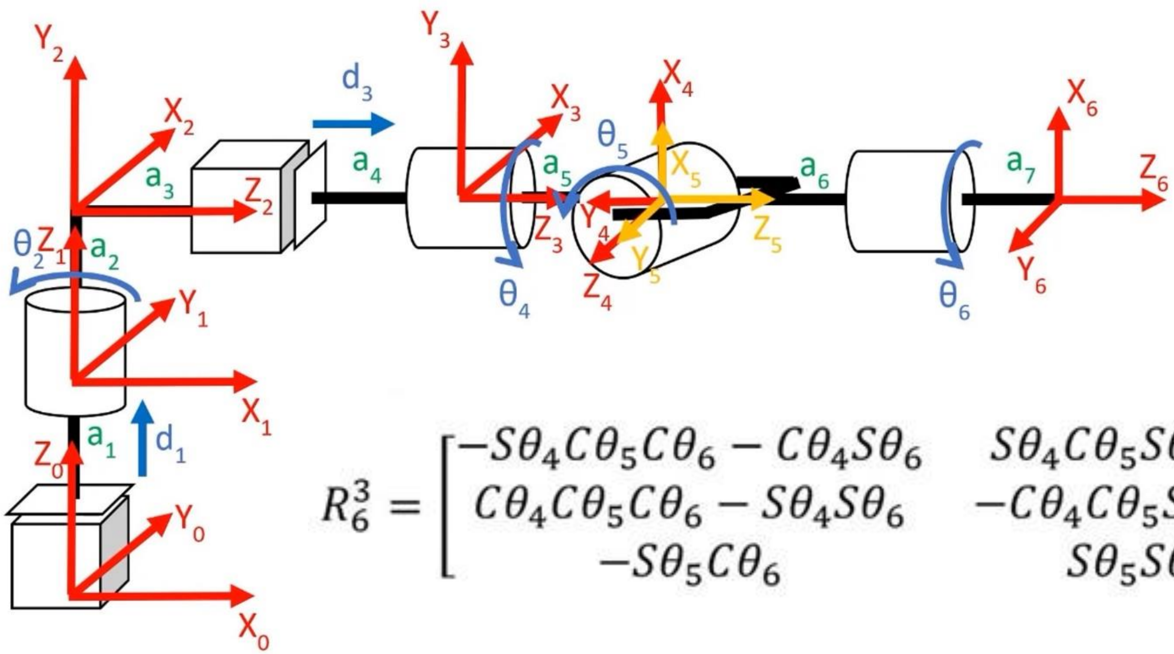
$$R_6^0 = R_3^0 R_6^3$$

$$R_3^{0^{-1}} R_6^0 = R_3^{0^{-1}} R_3^0 R_6^3$$

$$R_6^3 = R_3^{0^{-1}} R_6^0$$

6 DoF procedure break down

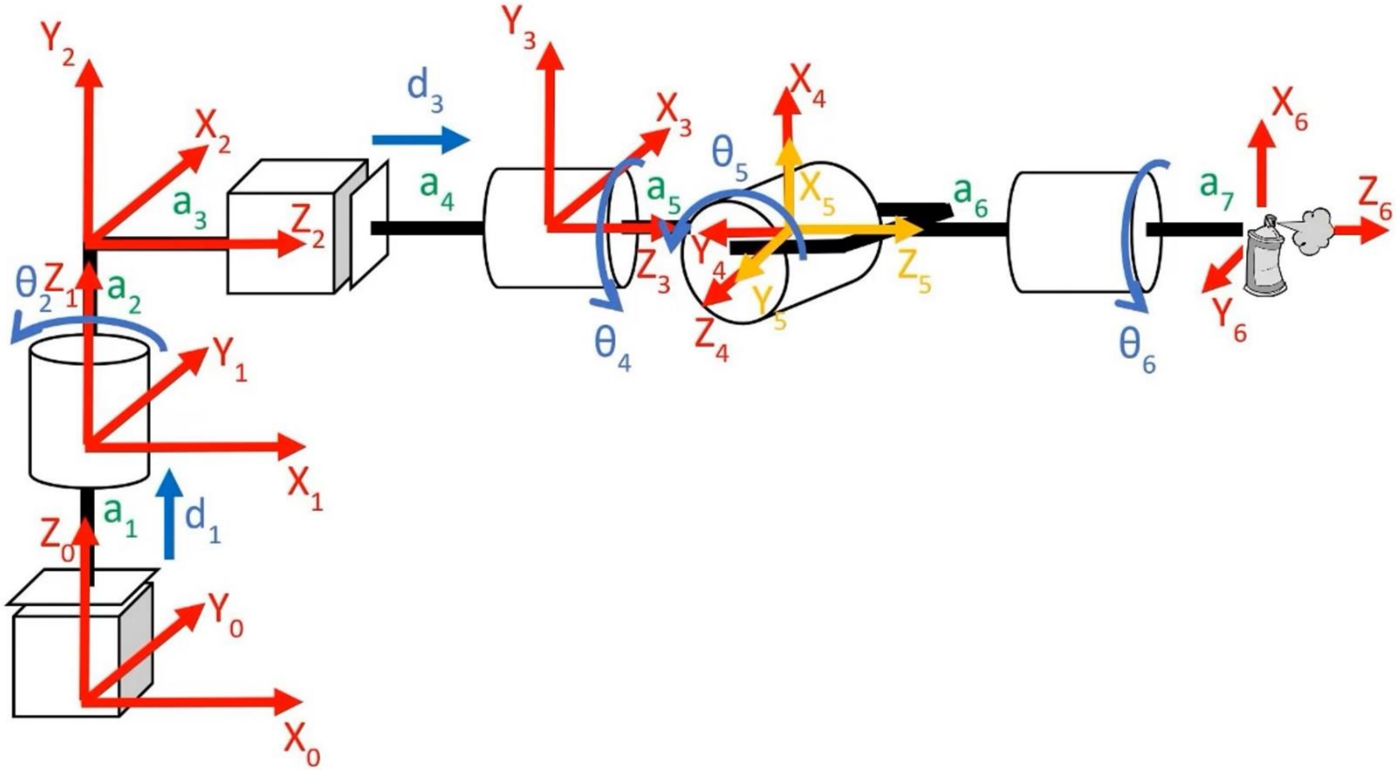
- **Step 04**: Do forward kinematics on the last three joints & pull out the rotation part R 3_6



$$R_6^3 = \begin{bmatrix} -S\theta_4 C\theta_5 C\theta_6 - C\theta_4 S\theta_6 & S\theta_4 C\theta_5 S\theta_6 - C\theta_4 C\theta_6 & -S\theta_4 S\theta_5 \\ C\theta_4 C\theta_5 C\theta_6 - S\theta_4 S\theta_6 & -C\theta_4 C\theta_5 S\theta_6 - S\theta_4 C\theta_6 & C\theta_4 S\theta_5 \\ -S\theta_5 C\theta_6 & S\theta_5 S\theta_6 + C\theta_5 & C\theta_5 \end{bmatrix}$$

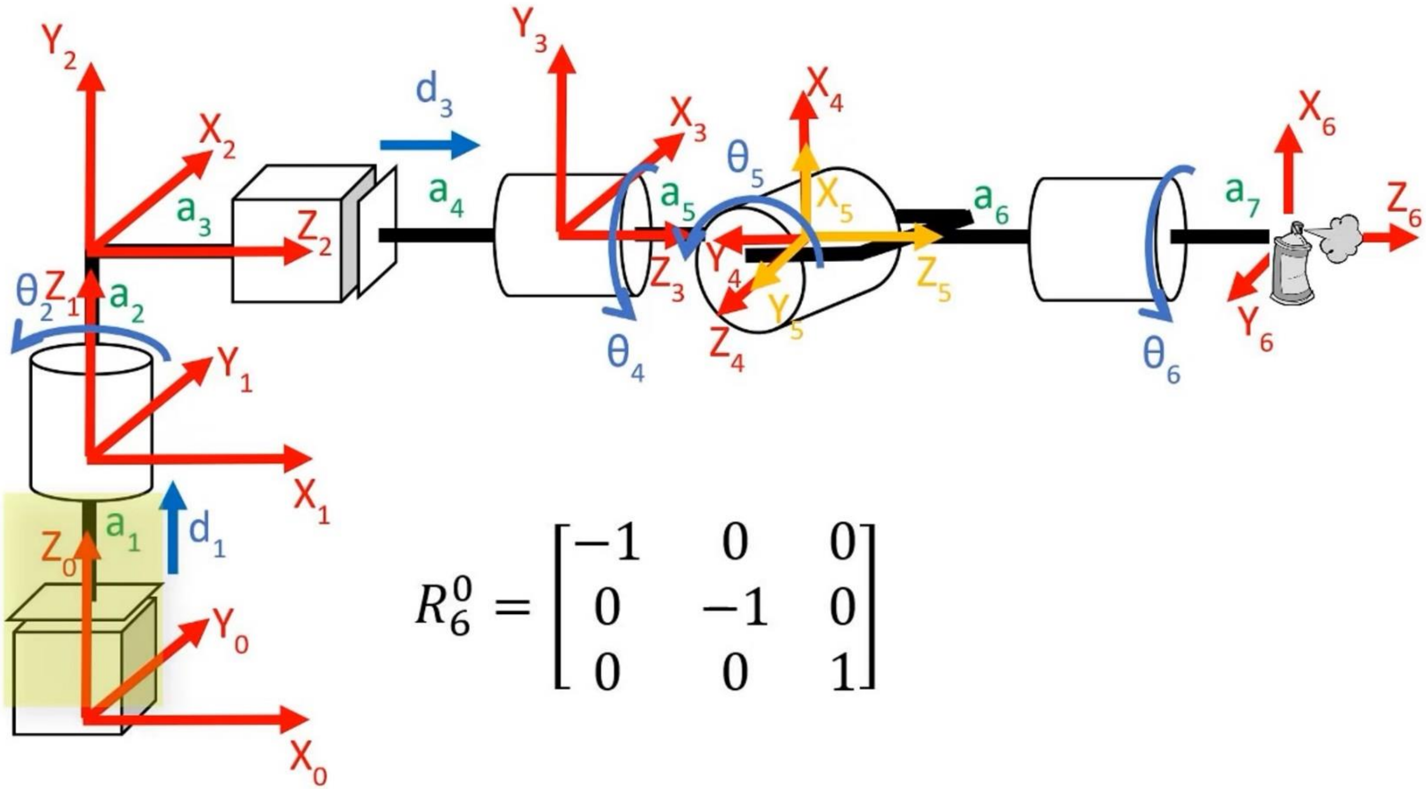
6 DoF procedure break down

- **Step 05**: Specify what you want the rotation matrix R 0_6 to be



6 DoF procedure break down

- **Step 05**: Specify what you want the rotation matrix R 0_6 to be
  - Create rotation matrix with short cut method for R 0_6



$$R_6^0 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

6 DoF procedure break down

- **Step 05**: Specify what you want the rotation matrix R 0_6 to be
  - Create rotation matrix with short cut method for R 0_6
- We can use any value of R 0_6 until it is a valid rotation matrix
  - A valid rotation matrix needs to have two properties
    1. Every row and column of the matrix needs to have a vector length of 1
    2. The matrix needs to describe a right hand coordinate frame. You can check a rotation matrix is a right hand coordinate frame by using shortcut method of writing rotation matrices in reverse or in other words draw one frame and figure out what the second frame could be using shortcut method to see if the second frame does or doesn't follow RHR

$$R_6^0 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \sqrt{(-1)^2 + (0)^2 + (0)^2} = 1$$

6 DoF procedure break down

- **Step 06**: Given a derived x, y and z position, solve for the first three joints using the inverse kinematics equations from step 01
- E.g if
  - X = 5.0
  - Y = 0.0
  - Theta2 = np.arctan2(Y, X)

6 DoF procedure break down

- **Step 07**: Plug in those variables (from Step 06) and use the rotation matrix to solve for the last three joints

$$R_6^0 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
R3_6 =  [[ 0. -1.  0.]
 [ 0.  0.  1.]
 [-1.  0.  0.]]
Theta5 =  1.5707963267948966  radians
Theta6 =  0.0  radians
Theta4 =  0.0  radians
R3_6_check =  [[-0.000000e+00 -1.000000e+00 -0.000000e+00]
 [ 6.123234e-17 -0.000000e+00  1.000000e+00]
 [-1.000000e+00  6.123234e-17  6.123234e-17]]
```

```
import numpy as np
X = 5.0
Y = 0.0

Theta2 = np.arctan2(Y, X)

R0_6 = [[-1.0, 0.0, 0.0],
                    [0.0, -1.0, 0.0],
                    [0.0, 0.0, 1.0]]

R0_3 = [[-np.sin(Theta2), 0.0, np.cos(Theta2)],   [np.cos(Theta2), 0.0, np.sin(Theta2)],  [0.0, 1.0, 0.0]]

invR0_3 = np.linalg.inv(R0_3)

R3_6 = np.dot(invR0_3, R0_6)

print ('R3_6 = ', np.matrix(R3_6))

Theta5 = np.arccos(R3_6[2][2])

print ('Theta5 = ', Theta5, ' radians')

Theta6 = np.arccos(-R3_6[2][0]/np.sin(Theta5))

print ('Theta6 = ', Theta6, ' radians')

Theta4 = np.arccos(R3_6[1][2]/np.sin(Theta5))

print ('Theta4 = ', Theta4, ' radians')

R3_6_check = [[-np.sin(Theta4)*np.cos(Theta5)*np.cos(Theta6) - np.cos(Theta4)*np.sin(Theta6) ,
np.sin(Theta4)*np.cos(Theta5)*np.sin(Theta6) - np.cos(Theta4)*np.cos(Theta6), -np.sin(Theta4)*np.sin(Theta5) ],
[np.cos(Theta4)*np.cos(Theta5)*np.cos(Theta6) - np.sin(Theta4)*np.sin(Theta6),
-np.cos(Theta4)*np.cos(Theta5)*np.sin(Theta6) - np.sin(Theta4)*np.cos(Theta6), np.cos(Theta4)*np.sin(Theta5)],
[-np.sin(Theta5)*np.cos(Theta6), np.sin(Theta5)*np.sin(Theta6)+np.cos(Theta5), np.cos(Theta5)]]

print ('R3_6_check = ', np.matrix(R3_6_check))
```

```
R3_6 =  [[ 0. -1.  0.]
 [ 0.  0.  1.]
 [-1.  0.  0.]]
Theta5 =  1.5707963267948966  radians
Theta6 =  0.0  radians
Theta4 =  0.0  radians
R3_6_check =  [[-0.000000e+00 -1.000000e+00 -0.000000e+00]
 [ 6.123234e-17 -0.000000e+00  1.000000e+00]
 [-1.000000e+00  6.123234e-17  6.123234e-17]]
```

End