Q1

There are infinite solutions, and we can use geometric approach to prove that.

Method1: Geometric proof

Firstly, we put an auxiliary Line between joint 1 and joint 4, naming the auxiliary link as l_{14}. Similarly, put another line between joint 2 and joint 4, naming it l_{24}. The angles between the auxiliary lines ad links are α_1 and α_2 (as is shown in figure 1).

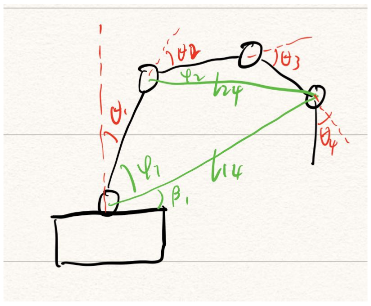


Figure 1: illustration picture

If we look at the triangle surrounded by l_2, l_3, l_{24}, we can find out the expression of l_{24} can be written as:

$$l_{24} = \sqrt{l_2^2 + l_3^2 + 2l_2l_3\,cos\,(\pi - \theta_3)}$$

$$= \sqrt{l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3)}$$

What's more,

$$\cos(\varphi_2) = \frac{l_2^2 + l_{24}^2 - l_3^2}{2l_2l_{24}}$$

$$= \frac{l_2^2 + l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3) - l_3^2}{2l_2\sqrt{l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3)}}$$

$$= \frac{l_2 + l_3\cos(\theta_3)}{\sqrt{l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3)}}$$

Hence

$$\varphi_2 = arccos\frac{l_2 + l_3\cos(\theta_3)}{\sqrt{l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3)}}$$

Similarly, for the triangle made up of l_1, l_{14}, l_{24}

$$\cos(\varphi_1) = \frac{l_{14}^2 + l_1^2 - l_{24}^2}{2l_1l_{14}}$$

$$= \frac{l_{14}^2 + l_1^2 - l_2^2 - l_3^2 - 2l_2l_3\cos(\theta_3)}{2l_1l_{14}}$$

Where l_{14} is the distance between end-effector and origin, so we can have

$$\cos(\varphi_1) = \frac{(P_{ex}^0)^2 + \left(P_{ey}^0\right)^2 + l_1^2 - l_2^2 - l_3^2 - 2l_2l_3\cos(\theta_3)}{2l_1\sqrt{(P_{ex}^0)^2 + \left(P_{ey}^0\right)^2}}$$

$$\varphi_1 = \arccos\frac{(P_{ex}^0)^2 + \left(P_{ey}^0\right)^2 + l_1^2 - l_2^2 - l_3^2 - 2l_2l_3\cos(\theta_3)}{2l_1\sqrt{(P_{ex}^0)^2 + \left(P_{ey}^0\right)^2}}$$

Assume the angle of l_{14} is β_1

$$\beta_1 = arctan\frac{P_{ey}^0}{P_{ex}^0}$$

Now we can find the expression for each angle

For θ_1

$$\theta_1 = \beta_1 \pm \varphi_1 - \frac{\pi}{2} = arctan\frac{P_{ey}^0}{P_{ex}^0} \pm \arccos\frac{\left(P_{ex}^0\right)^2 + \left(P_{ey}^0\right)^2 + l_1^2 - l_2^2 - l_3^2 - 2l_2l_3\cos(\theta_3)}{2l_1\sqrt{\left(P_{ex}^0\right)^2 + \left(P_{ey}^0\right)^2}} - \frac{\pi}{2}$$

(+ if $\varphi_1 + \theta_2 < 0$, - if $\varphi_1 + \theta_2 > 0$)

For θ_2

In the triangle surrounded by l_1, l_{14}, l_{24}, we have the relationship that

$$\cos(\pi - \theta_2 - \alpha_2) = \frac{l_{24}^2 + l_1^2 - l_{14}^2}{2l_1l_{24}}$$

Sub $l_{14} = \sqrt{(P_{ex}^0)^2 + \left(P_{ey}^0\right)^2}$, $l_{24} = \sqrt{l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3)}$,into it

$$\cos(\pi - \theta_2 - \varphi_2) = \frac{l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3) + l_1^2 - (P_{ex}^0)^2 - \left(P_{ey}^0\right)^2}{2l_1\sqrt{l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3)}}$$

$$\cos(\theta_2 + \varphi_2) = -\frac{l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3) + l_1^2 - (P_{ex}^0)^2 - \left(P_{ey}^0\right)^2}{2l_1\sqrt{l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3)}}$$

Hence

$$\theta_2 = arccos\frac{-l_2^2 - l_3^2 - 2l_2l_3\cos(\theta_3) - l_1^2 + (P_{ex}^0)^2 + \left(P_{ey}^0\right)^2}{2l_1\sqrt{l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3)}} - arccos\frac{l_2 + l_3\cos(\theta_3)}{\sqrt{l_2^2 + l_3^2 + 2l_2l_3\cos(\theta_3)}}$$

For θ_4, if we have a γ to represent an expected pose

$$\theta_4 = \gamma - \theta_1 - \theta_2 - \theta_3$$

For θ_3, we can treat it as a variable where it obeys the inequality of triangle

$$l_1 + l_{24} \geq l_{14}$$

$$l_1 + \sqrt{l_2^2 + l_3^2 + 2l_2 l_3 \cos(\theta_3)} \geq \sqrt{(P_{ex}^0)^2 + \left(P_{ey}^0\right)^2}$$

$$\cos(\theta_3) \geq \frac{(\sqrt{(P_{ex}^0)^2 + \left(P_{ey}^0\right)^2} - l_1)^2 - l_2^2 - l_3^2}{2l_2 l_3}$$

Since θ_3 has infinite solutions and $\theta_1, \theta_2, \theta_4$ are expressed by θ_3, we can conclude that the system has infinite solutions.

Only one exception occurs when the position of the joint right before end effector can only be reached by flatten link1, link2 and link3, which means $l_1 + l_2 + l_3 = l_{14}$

Method 2: Logical deduction

There are infinite solutions

Proof:

For a 2D 4R-planar manipulator whose end effector has been given, it means we can have a given q_e

$$q_e = \begin{bmatrix} r_{e1} \\ r_{e2} \\ r_{e3} \\ p_{e1} \\ p_{e2} \\ p_{e3} \end{bmatrix}$$

According to the DH of the manipulator, we can compute out the third joint's position:

$$q_3 = \begin{bmatrix} r_{e1} - l_4 \cos(p_{e1}) \\ r_{e2} - l_4 \cos(p_{e2}) \\ r_{e3} - l_4 \cos(p_{e3}) \\ p_{31} \\ p_{32} \\ p_{33} \end{bmatrix}$$

Where r_{31}, r_{32} *and* r_{33} are determined and p_{31}, p_{32} and p_{33} are unknown.

Therefore, the proof is equivalent to prove q_3 has infinite solutions.

While p_{31}, p_{32} and p_{33} has infinite effective combinations, we can say if there is an infinite number of different combinations that are valid, a 2D 4R-planar manipulator with a given end effector would have infinite pose solutions.

According to the solution equation for 3Rmanipulator in lecture 6 [1], infinite number of different combinations exist.

Therefore, a 4R-planar manipulator has infinite inverse kinematic solution

Q2:

Criteria:

1. Physical constrains. For instance, when a joint can revote from -45 degrees to 45 degrees, even we have a solution outside the range that make the whole robot work perfectly, we cannot adopt it.
2. Minimum effort cost. If we got multiple inverse kinematic solutions, we could choose the one that cost the robot minimum effort to reach that pose. By this, probability of machine failure is cutting down and machine life is extended.
3. Self-collision. The links have a collision volume with each other and we need to avoid it.

Q3:

atan and **atan2** both return the arctangent of x and y/x, respectively. There are two main differences.

- The output ranges. atan() returns a value in the range of $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ radians, while atan2() returns a value in range of $-\pi$ to π.
- Singularity. When the input of atan() has a x=0, then the atan2() will drop an error

Q4:

a. In order to figure out the Jacobian matrix for YouBot manipulator, considering we already have the Jacobian matrix expression below, all we need to do is substituting the YouBot parameters into the expression below.

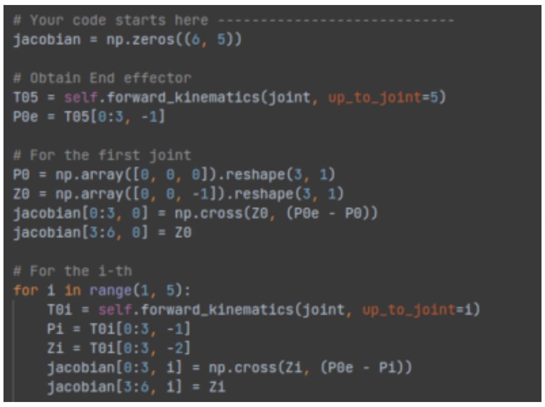
$$J = \begin{bmatrix} J_{P_1} & J_{P_2} & \cdots & J_{P_n} \\ J_{O_1} & J_O & \cdots & J_{O_1} \end{bmatrix}$$

Where:

$$J_{P_i} = \begin{cases} z_{i-1}^0 \times (p_e^0 - o_{i-1}^0), & if\ joint\ i\ is\ revolute \\ z_{i-1}^0, & if\ joint\ i\ is\ rismatic \end{cases}$$

$$J_{O_i} = \begin{cases} z_{i-1}^0, & if\ joint\ i\ is\ revolute \\ 0, & if\ joint\ i\ is\ rismatic \end{cases}$$

In this case, we firstly are supposed to obtain transfermation of end effector T05 (As is shown in figure 2) and then extract the coordinates of origin of end effector since the first three elements of last column represent the coordinates of corresponding coordinates according to lecture 5 (shown in figure 3)[2]. Then apply Jacobian matrix expression to the parameters. Note that, for the first joint, z0 should be set to [0,0,-1] to match the URDF.

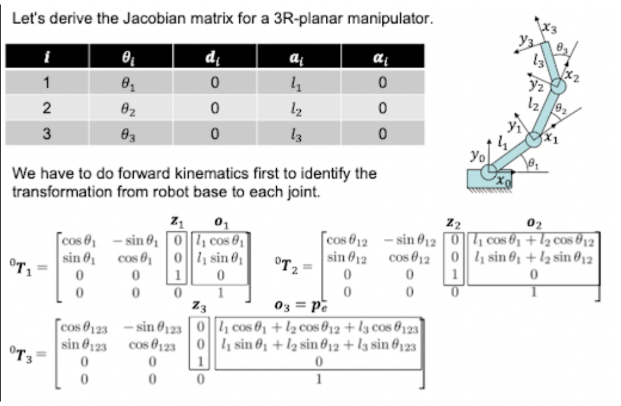


```python
# Your code starts here ----------------------------
jacobian = np.zeros((6, 5))

# Obtain End effector
T05 = self.forward_kinematics(joint, up_to_joint=5)
P0e = T05[0:3, -1]

# For the first joint
P0 = np.array([0, 0, 0]).reshape(3, 1)
Z0 = np.array([0, 0, -1]).reshape(3, 1)
jacobian[0:3, 0] = np.cross(Z0, (P0e - P0))
jacobian[3:6, 0] = Z0

# For the i-th
for i in range(1, 5):
    T0i = self.forward_kinematics(joint, up_to_joint=i)
    Pi = T0i[0:3, -1]
    Zi = T0i[0:3, -2]
    jacobian[0:3, i] = np.cross(Zi, (P0e - Pi))
    jacobian[3:6, i] = Zi
```



Let's derive the Jacobian matrix for a 3R-planar manipulator.

i	θ_i	d_i	a_i	α_i
1	θ_1	0	l_1	0
2	θ_2	0	l_2	0
3	θ_3	0	l_3	0

We have to do forward kinematics first to identify the transformation from robot base to each joint.

$$ {}^0\boldsymbol{T}_1 = \begin{array}{cccc} & & \boldsymbol{z}_1 & \boldsymbol{o}_1 \\ \cos\theta_1 & -\sin\theta_1 & 0 & l_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & l_1\sin\theta_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \qquad {}^0\boldsymbol{T}_2 = \begin{array}{cccc} & & \boldsymbol{z}_2 & \boldsymbol{o}_2 \\ \cos\theta_{12} & -\sin\theta_{12} & 0 & l_1\cos\theta_1 + l_2\cos\theta_{12} \\ \sin\theta_{12} & \cos\theta_{12} & 0 & l_1\sin\theta_1 + l_2\sin\theta_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} $$

$$ {}^0\boldsymbol{T}_3 = \begin{array}{cccc} & & \boldsymbol{z}_3 & \boldsymbol{o}_3 = \boldsymbol{p}_e \\ \cos\theta_{123} & -\sin\theta_{123} & 0 & l_1\cos\theta_1 + l_2\cos\theta_{12} + l_3\cos\theta_{123} \\ \sin\theta_{123} & \cos\theta_{123} & 0 & l_1\sin\theta_1 + l_2\sin\theta_{12} + l_3\sin\theta_{123} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} $$

Figure 2: code of cw2q4a figure 3: relationship between transfermation and origin

b.

The D-H table for YouBot manipulator:

Joint	a	α	d	θ
1	a_1	$\frac{\pi}{2}$	d_1	θ_1
2	a_2	0	0	θ_2
3	a_3	0	0	θ_3
4	0	$\frac{\pi}{2}$	0	θ_4
5	0	0	d_5	θ_5

Also, as we have a full transformation function:

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

And the chain rule:

$$ {}^0T_n = {}^0T_1\, {}^1T_2 \ldots {}^{n-1}T_n $$

Therefore, we can compute the transformation function of joint 0 to end effector:

$$ {}^0T_n = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_{ex}^0 \\ r_{21} & r_{22} & r_{23} & p_{ey}^0 \\ r_{31} & r_{32} & r_{33} & p_{ez}^0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Where

$$r_{11} = \sin(\theta_1) * \sin(\theta_5) - \cos(\theta_5) \\ * \big(\cos(\theta_4) * (\cos(\theta_1) * \sin(\theta_2) * \sin(\theta_3) - \cos(\theta_1) * \cos(\theta_2) * \cos(\theta_3)) \\ + \sin(\theta_4) * (\cos(\theta_1) * \cos(\theta_2) * \sin(\theta_3) + \cos(\theta_1) * \cos(\theta_3) * \sin(\theta_2))\big) \\ = sin\theta_1 \sin(\theta_5) + cos\theta_1 cos\theta_{1234} cos\theta_5$$

$$r_{12} = \cos(\theta_5) * \sin(\theta_1) + \sin(\theta_5) \\ * \big(\cos(\theta_4) * (\cos(\theta_1) * \sin(\theta_2) * \sin(\theta_3) - \cos(\theta_1) * \cos(\theta_2) * \cos(\theta_3)) \\ + \sin(\theta_4) * (\cos(\theta_1) * \cos(\theta_2) * \sin(\theta_3) + \cos(\theta_1) * \cos(\theta_3) * \sin(\theta_2))\big) \\ = \cos(\theta_5) * \sin(\theta_1) + cos\theta_1 cos\theta_{1234} sin\theta_5$$

$$r_{13} = \cos(\theta_4) * (\cos(\theta_1) * \cos(\theta_2) * \sin(\theta_3) + \cos(\theta_1) * \cos(\theta_3) * \sin(\theta_2)) - \sin(\theta_4) \\ * (\cos(\theta_1) * \sin(\theta_2) * \sin(\theta_3) - \cos(\theta_1) * \cos(\theta_2) * \cos(\theta_3)) \\ = cos\theta_1 sin\theta_{234}$$

$$r_{21} = -\cos(\theta_1) * \sin(\theta_5) - \cos(\theta_5) \\ * \big(\cos(\theta_4) * (\sin(\theta_1) * \sin(\theta_2) * \sin(\theta_3) - \cos(\theta_2) * \cos(\theta_3) * \sin(\theta_1)) \\ + \sin(\theta_4) * (\cos(\theta_2) * \sin(\theta_1) * \sin(\theta_3) + \cos(\theta_3) * \sin(\theta_1) * \sin(\theta_2))\big) \\ = -cos\theta_1 * \sin(\theta_5) - \sin\theta_1 \cos\theta_{234} \cos\theta_5$$

$$r_{22} = \sin(\theta_5) * \big(\cos(\theta_4) * (\sin(\theta_1) * \sin(\theta_2) * \sin(\theta_3) - \cos(\theta_2) * \cos(\theta_3) * \sin(\theta_1)) \\ + \sin(\theta_4) * (\cos(\theta_2) * \sin(\theta_1) * \sin(\theta_3) + \cos(\theta_3) * \sin(\theta_1) * \sin(\theta_2))\big) \\ - \cos(\theta_1) * \cos(\theta_5) = -cos\theta_1 * \cos(\theta_5) - \sin\theta_1 \cos\theta_{234} \sin\theta_5$$

$$r_{23} = \cos(\theta_4) * (\cos(\theta_2) * \sin(\theta_1) * \sin(\theta_3) + \cos(\theta_3) * \sin(\theta_1) * \sin(\theta_2)) - \sin(\theta_4) \\ * (\sin(\theta_1) * \sin(\theta_2) * \sin(\theta_3) - \cos(\theta_2) * \cos(\theta_3) * \sin(\theta_1)) \\ = sin\theta_1 sin\theta_{234}$$

$$r_{31} = \cos(\theta_5) * \big(\cos(\theta_4) * (\cos(\theta_2) * \sin(\theta_3) + \cos(\theta_3) * \sin(\theta_2)) + \sin(\theta_4) \\ * (\cos(\theta_2) * \cos(\theta_3) - \sin(\theta_2) * \sin(\theta_3))\big) = cos\theta_5 sin\theta_{234}$$

$$r_{32} = -\sin(\theta_5) \\ * \big(\cos(\theta_4) * (\cos(\theta_2) * \sin(\theta_3) + \cos(\theta_3) * \sin(\theta_2)) + \sin(\theta_4) \\ * (\cos(\theta_2) * \cos(\theta_3) - \sin(\theta_2) * \sin(\theta_3))\big) = -sin\theta_5 sin\theta_{234}$$

$$r_{33} = \sin(\theta_4) * (\cos(\theta_2) * \sin(\theta_3) + \cos(\theta_3) * \sin(\theta_2)) - \cos(\theta_4) * (\cos(\theta_2) * \cos(\theta_3) - \sin(\theta_2) * \sin(\theta_3)) = -cos\theta_{234}$$

$$p_{ex}^0 = d5 * \big(\cos(\theta_4) * (\cos(\theta_1) * \cos(\theta_2) * \sin(\theta_3) + \cos(\theta_1) * \cos(\theta_3) * \sin(\theta_2)) - \sin(\theta_4) * (\cos(\theta_1) * \sin(\theta_2) * \sin(\theta_3) - \cos(\theta_1) * \cos(\theta_2) * \cos(\theta_3))\big) + a1 * \cos(\theta_1) + a2 * \cos(\theta_1) * \cos(\theta_2) + a3 * \cos(\theta_1) * \cos(\theta_2) * \cos(\theta_3) - a3 * \cos(\theta_1) * \sin(\theta_2) * \sin(\theta_3)$$

$$= d5\sin(\theta_{234})\cos(\theta_1) + a1\cos(\theta_1) + a2\cos(\theta_1)\cos(\theta_2) + a3\cos(\theta_1)\cos(\theta_{23}) + a4\cos(\theta_1)\cos(\theta_{234})$$

$$p_{ey}^0 = d5 * (cos\ (\theta_4) * (cos\ (\theta_2) * sin\ (\theta_1) * sin\ (\theta_3) + cos\ (\theta_3) * sin\ (\theta_1) * sin\ (\theta_2)\ ) - sin\ (\theta_4) * (sin\ (\theta_1) * sin\ (\theta_2) * sin\ (\theta_3) - cos\ (\theta_2) * cos\ (\theta_3) * sin\ (\theta_1)\ )) + a1 * sin\ (\theta_1) + a2 * cos\ (\theta_2) * sin\ (\theta_1) + a3 * cos\ (\theta_2) * cos\ (\theta_3) * sin\ (\theta_1) - a3 * sin\ (\theta_1) * sin\ (\theta_2) * sin\ (\theta_3)$$

$$= d5\sin(\theta_{234})\cos(\theta_1) + a1\sin(\theta_1) + a2\sin(\theta_1)\cos(\theta_2) + a3\sin(\theta_1)\cos(\theta_{23}) + a4\sin(\theta_1)\cos(\theta_{234})$$

$$p_{ez}^0 = d1 + a2 * \sin(\theta_2) - d5 * \big(\cos(\theta_4) * (\cos(\theta_2) * \cos(\theta_3) - \sin(\theta_2) * \sin(\theta_3)) - \sin(\theta_4) * (\cos(\theta_2) * \sin(\theta_3) + \cos(\theta_3) * \sin(\theta_2))\big) + a3 * \cos(\theta_2) * \sin(\theta_3) + a3 * \cos(\theta_3) * \sin(\theta_2)$$

$$= d1 + a2\sin(\theta_2) + a3\sin(\theta_{23}) + a4\sin(\theta_{234}) - d5\cos(\theta_{234})$$

In position x and y we can find that

$r_{11} = sin\theta_1 \sin(\theta_5) + cos\theta_1 cos\theta_{1234} cos\theta_5$

$r_{12} = \cos(\theta_5)\sin(\theta_1) + cos\theta_1 cos\theta_{1234} sin\theta_5$

$r_{13} = cos\theta_1 sin\theta_{234}$

$r_{21} = -cos\theta_1 * \sin(\theta_5) - \sin\theta_1 \cos\theta_{234} \cos\theta_5$

$r_{22} = -cos\theta_1 * \cos(\theta_5) - \sin\theta_1 \cos\theta_{234} \sin\theta_5$

$r_{23} = sin\theta_1 sin\theta_{234}$

$r_{31} = cos\theta_5 sin\theta_{234}$

$r_{32} = -sin\theta_5 sin\theta_{234}$

$r_{33} = -cos\theta_{234}$

Solve them we have

$$\theta_1 = arcsin\frac{r_{23}}{\sqrt{1 - {r_{33}}^2}}$$

$$\theta_{234} = \arccos(-r_{33})$$

$$\theta_5 = \arctan\left(-\frac{r_{32}}{r_{31}}\right)$$

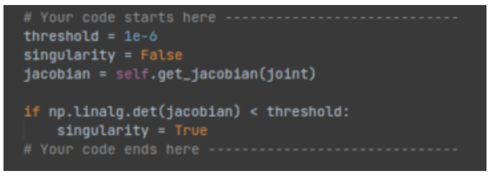
Then we have a set of equations, which is

$$\begin{cases} \theta_1 = arcsin\frac{r_{23}}{\sqrt{1-{r_{33}}^2}} \\ \theta_{234} = \arccos\left(-r_{33}\right) \\ \theta_5 = \arctan\left(-\frac{r_{32}}{r_{31}}\right) \\ p_{ex}^0 = \cos\left(\theta_1\right)(d5\sin(\theta_{234}) + a1\cos(\theta_1) + a2\cos(\theta_2) + a3\cos(\theta_{23}) + a4\cos(\theta_{234})) \\ p_{ey}^0 = \sin(\theta_1)\left(d5\sin(\theta_{234}) + a1\cos(\theta_1) + a2\cos(\theta_2) + a3\cos(\theta_{23}) + a4\cos(\theta_{234})\right) \\ p_{ez}^0 = d1 + a2\sin(\theta_2) + a3\sin(\theta_{23}) + a4\sin(\theta_{234}) - d5\cos(\theta_{234}) \end{cases}$$

The rank of the equation set < the number of variables, which means it has infinite solutions. Therefore, this set may be the solutions.

c.

According to lecture 5, singularity can be found by calculating the determinates the Jacobian matrix and the determinates cannot be 0. Therefore, a simple 0 checking algorithm can be applied to detect singularity. The code is shown in figure 4.



```
# Your code starts here ----------------------------
threshold = 1e-6
singularity = False
jacobian = self.get_jacobian(joint)

if np.linalg.det(jacobian) < threshold:
    singularity = True
# Your code ends here ------------------------------
```

Figure 4: code of cw2q4c

Q5:

a.

1. The drive system can be obtained by controlling each wheel's voltage to control the velocity of wheels and hence, control the movement of robot. The robot turns right if $V_L > V_R$, turns left if $V_L < V_R$ and goes straight if $V_L = V_R$.
2. Non-holonomic. According to the introduction from [3], we know that only the total degrees of freedom equal to the controllable degrees of freedom, will a robot be described as holonomic. In this case, the robot car has two axis freedom and one on its orientation, but we can only control its orientation and moving forward and backward. Side movement cannot be achieved.
3. Configuration space means the set of all possible configuration the robot can reach to.

b.

1. Firstly, determine configuration space Q, configuration space obstacle Qo.
   Then, basing on the, we could construct a configuration space roadmap, which is a network of curves that represents Qfree.
   Finally, with roadmap, we can find the closet path following their order.
2. If order did not matter, we can firstly find the closest path between each two checkpoints on roadmap
   Then, using Dijkstra algorithm to find closest path.

c.

I would use a fourth-order polynomials, as it can make the robot has a smooth acceleration curve which can avoid robot's falling over.

Which means:

$$q_s(0) = \dot{q}_s(0) = \ddot{q}_s(0) = 0$$
$$q_f(t_f) = \dot{q}_f(t_f) = \ddot{q}_f(t_f) = 0$$

Other points have their corresponding non-zero parameters.

And the polynomials are:

$$q(t) = \begin{cases} Q_1(t) = a_{10} + a_{11}t + a_{12}t^2 + a_{13}t^3 + a_{14}t^4, t_s \le t < t_{p1} \\ Q_2(t) = a_{20} + a_{21}t + a_{22}t^2 + a_{23}t^3 + a_{24}t^4, t_{p1} \le t < t_{p2} \\ Q_3(t) = a_{30} + a_{31}t + a_{32}t^2 + a_{33}t^3 + a_{34}t^4, t_{p2} \le t < t_{p3} \\ Q_4(t) = a_{40} + a_{41}t + a_{42}t^2 + a_{43}t^3 + a_{44}t^4, t_{p3} \le t < t_{p4} \\ Q_5(t) = a_{50} + a_{51}t + a_{52}t^2 + a_{53}t^3 + a_{54}t^4, t_{p4} \le t < t_f \end{cases}$$

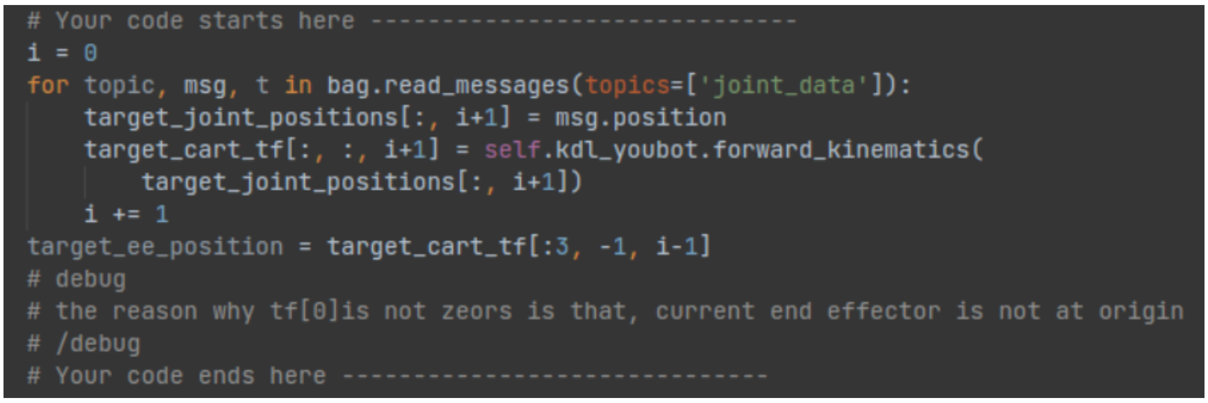
d.

missing water pick-up only occurs when the robot would like to change its movement curves, which means the velocity difference of two wheels changed.
Then it can be solved by making the robot spin a full circle as long as it wants to change the robot wheel's speed-difference.

e.

Firstly, mapping the floor to 480*480 grids with default state as "0" and mark all the obstacles and "No go zone" onto the grids with changing corresponding grids' states to "1".
Then, set ordered check points to both terminals of each row and the number of rows equal to 480/20, which could make the robot move from row to row until covering the whole floor.
Finally, apply path planning algorithm to find the right path.

Q6.a.
Following the tutorial from ROS bagfile Wiki [4], we can easily extract messages in the bag file with function bag.read_message(). In figure 5, the data extracted from bag file is stored in variable "target_joint_positions" and corresponding transformations are stored in "target_cart_tf". The target position of end effector is the first three elements in the top-right corner of the final transformation.

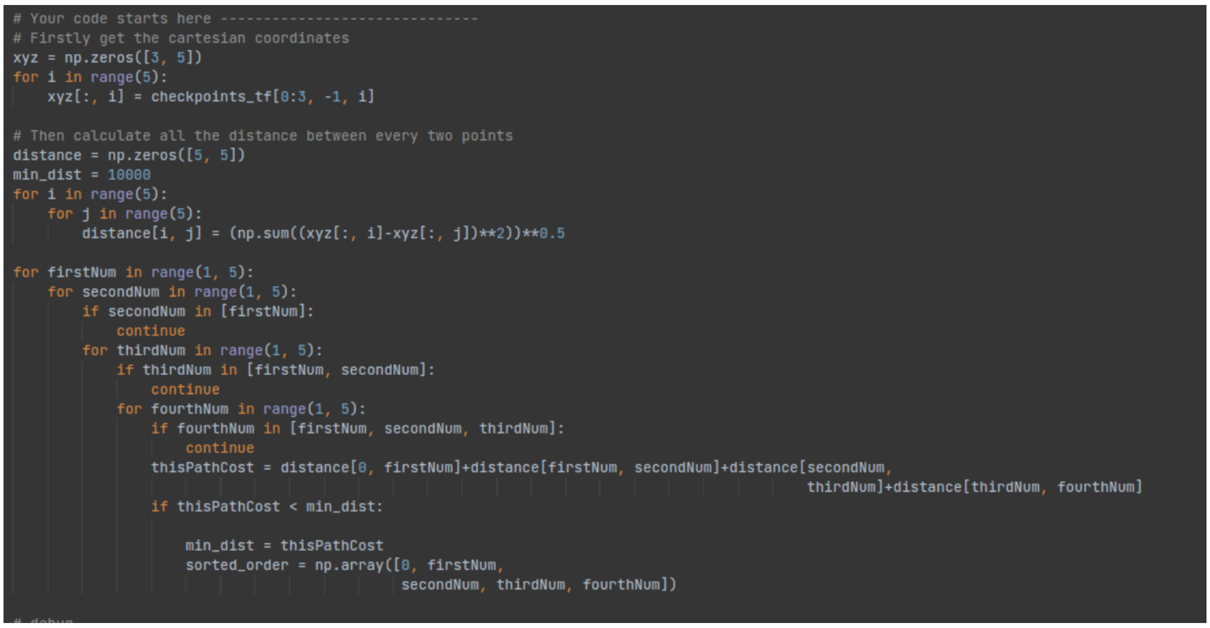


```python
# Your code starts here ------------------------------
i = 0
for topic, msg, t in bag.read_messages(topics=['joint_data']):
    target_joint_positions[:, i+1] = msg.position
    target_cart_tf[:, :, i+1] = self.kdl_youbot.forward_kinematics(
        target_joint_positions[:, i+1])
    i += 1
target_ee_position = target_cart_tf[:3, -1, i-1]
# debug
# the reason why tf[0]is not zeors is that, current end effector is not at origin
# /debug
# Your code ends here ------------------------------
```

Figure 5: code for cw2q6a

b.

In order to figure out the shortest path, we should firstly get the cartesian coordinates of all checkpoints and then calculate the distance of each two points. Using the distances, we could go through all potential path and compare each of them with the current shortest path until the end of the iteration. At the end the shortest path would be found.



```python
# Your code starts here ------------------------------
# Firstly get the cartesian coordinates
xyz = np.zeros([3, 5])
for i in range(5):
    xyz[:, i] = checkpoints_tf[0:3, -1, i]

# Then calculate all the distance between every two points
distance = np.zeros([5, 5])
min_dist = 10000
for i in range(5):
    for j in range(5):
        distance[i, j] = (np.sum((xyz[:, i]-xyz[:, j])**2))**0.5

for firstNum in range(1, 5):
    for secondNum in range(1, 5):
        if secondNum in [firstNum]:
            continue
        for thirdNum in range(1, 5):
            if thirdNum in [firstNum, secondNum]:
                continue
            for fourthNum in range(1, 5):
                if fourthNum in [firstNum, secondNum, thirdNum]:
                    continue
                thisPathCost = distance[0, firstNum]+distance[firstNum, secondNum]+distance[secondNum,
                                                                                   thirdNum]+distance[thirdNum, fourthNum]
                if thisPathCost < min_dist:

                    min_dist = thisPathCost
                    sorted_order = np.array([0, firstNum,
                                             secondNum, thirdNum, fourthNum])
```

Figure 6: code for cw2q6b

c.

To find the intermediate points, we could use the decoupled rotation and translation method:

$$p(t) = p_s + t(p_f - p_s)$$

$$R(t) = R_s * e^{\log(R_s^{-1} R_f)t}$$

and we could function $intermediate\_tfs$ to control this process, feeding transformations and checkpoints to function $decoupled\_rot\_and\_trans$. Note that, as the shape of variable $full\_checkpoint\_tfs$ is 4x4x(4xnum_points+5) we should insert generated points properly into checkpoints' intervals. So, we should put checkpoints to $i * (num\_points + 1)$ and other intermediates are supposed to put in range $i * (num_{points} + 1) + 1 : (i + 1)(num_{points} + 1)$.

d.

The inverse kinematic basing on Jacobian matrix can be implemented by iteration.

- Firstly, determine the parameters which are step size $\alpha = 0.2$, tolerance $r = 0.01$ and maximum iteration $i = 500$. Maximum iteration is set to avoid when steady state is higher than tolerance.
- Secondly, we should compute Jacobian matrix with built-in function $self.kdl\_youbot.get\_jacobian()$ and compute the current position using $self.kdl\_youbot.forward\_kinematics()$.
- Then compute the error between the target and current pose
- In the end check if error is in tolerate range or not.

Reference

[1] Agostino Stilli, "COMP0127-Lecture 6 Inverse Kinematics"

[2] Agostino Stilli, "COMP0127-Lecture 5 Jacobians and Differential Kinematics"

[3] "Robot Locomotion", [Online] Available:

http://www.robotplatform.com/knowledge/Classification_of_Robots/Holonomic_and_Non-Holonomic_drive.html

[4] "Bags Documentation" [Online] Available: http://wiki.ros.org/Bags

[5] Agostino Stilli, "COMP0127-Lecture 7 Path and Trajectory Planning"