COMS4733 Assignment02

Qianrui Zhao - qz2338

February 2019

Problem 01

(a) There are infinite solutions existing if we don't care about the angle. For any θ_1, there is a θ_2 which can makes the end effector point to a specific point. Moreover, the number of solutions is not depend on l_1 and l_2. Since the workspace for this laser arm is unlimited, no matter what length l_1 and l_2 is.

(b) According to Fig:

$$\phi = \theta_1 + \theta_2 \tag{1}$$

$$p_x = l_1 \cdot c_1 + (l_2 + d_3) \cdot c_\phi \tag{2}$$

$$p_y = l_1 \cdot s_1 + (l_2 + d_3) \cdot s_\phi \tag{3}$$

From equations (2) and (3), we can get:

$$l_1 \cdot c_1 = p_x - (l_2 + d_3) \cdot c_\phi \tag{4}$$

$$l_1 \cdot s_1 = p_y - (l_2 + d_3) \cdot s_\phi \tag{5}$$

Since ${s_1}^2 + {c_1}^2 = 1$:

$$l_1 = (p_x - (l_2 + d_3) \cdot cos_\phi)^2 + (p_y - (l_2 + d_3) \cdot sin_\phi)^2 \tag{6}$$

From (6), we can get the value of d_3, then it is very easy to compute θ_1 from equation (2), and $\theta_2 = \phi - \theta_1$

(c) If ϕ is specified, there will be 2 solutions if p is inside workspace, which depends on l_1. As illustrated in Fig 1, the workspace of l_1 is determined by its length. When If the line defined by ϕ and p is inside l_1's workspace, there are 2 solutions as drawed in orange line. When the line is right cross the circle workspace, only one solution exists. If the line is out of l_1's workspace, then there is no way for the laser to reach desired p, as shown in green line.

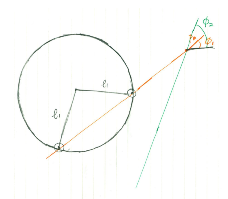

Figure 1: Workspace of joint 1 and given p and ϕ

Problem 02

(a) According to equation mentioned in lecture, $J(p)$ can be obtained by derivating p_x, p_y, p_y on joint variables:

$$J(p) = \begin{bmatrix} -(l_1 + l_2c_2 + l_3c_{23})c_1 & -l_2c_1s_2 - l_3c_1s_{23} & -l_3c_1s_{23} \\ (l_1 + l_2c_2 + l_3c_{23})c_1 & -l_2s_1s_2 - l_3s_1s_{23} & -l_3s_1s_{23} \\ 0 & l_2c_2 + l_3c_{23} & l_3c_{23} \end{bmatrix} \tag{1}$$

Substituting equation (1) with $\theta_1 = \theta_2 = \theta_3 = 0$, we can get:

$$J(p) = \tag{2}$$

Because the first joint is revolute:

$$z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{3}$$

To obtain z_1 and z_2, we build DH frames as shown in Fig, and get DH table:

Link	a_i	α_i	d_i	θ_i
1	l_1	90	0	θ_1
2	l_2	0	0	θ_2
3	l_3	0	0	θ_3

And we can get:

$$A_1^0 = \begin{bmatrix} c_1 & 0 & s_1 & l_1c_1 \\ s_1 & 0 & -c_1 & l_1s_1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$
A_2^0 = A_1^0 \times A_2^1 = \begin{bmatrix} c_1c_2 & -c_1s_2 & s_2 & l_2c_1c_2 + l_1c_1 \\ s_1c_2 & -s_1s_2 & -c_1 & l_2c_2s_1 + l_1s_1 \\ s_2 & c_2 & 0 & l_2s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}
$$

From the third column of (4) and (5):

$$
z_1 = \begin{bmatrix} s_1 \\ c_1 \\ 0 \end{bmatrix} \tag{6}
$$

$$
z_2 = \begin{bmatrix} s_1 \\ c_1 \\ 0 \end{bmatrix} \tag{7}
$$

Thus,

$$
J = \begin{bmatrix} -(l_1 + l_2c_2 + l_3c_{23})c_1 & -l_2c_1s_2 - l_3c_1s_{23} & -l_3c_1s_{23} \\ (l_1 + l_2c_2 + l_3c_{23})c_1 & -l_2s_1s_2 - l_3s_1s_{23} & -l_3s_1s_{23} \\ 0 & l_2c_2 + l_3c_{23} & l_3c_{23} \\ 0 & s_1 & s_1 \\ 0 & -c_1 & -c_1 \\ 1 & 0 & 0 \end{bmatrix} \tag{8}
$$

(b) Since the singularity occurs when $det(J_P) = 0$. According to previous part:

$$
det(J_P) = -l_2l_3(l_1s_3 - l_3s_2 + l_2c_2s_3 + l_3c_3^2s_3 + l_3c_2c_3s_3)
$$

Thus, singularity exists when:

$$
-l_2l_3(l_1s_3 - l_3s_2 + l_2c_2s_3 + l_3c_3^2s_3 + l_3c_2c_3s_3) = 0
$$

(c)
- Elbow happens when $\theta_3 = 0$ or $\theta_3 = \Pi$, as shown in Fig2.

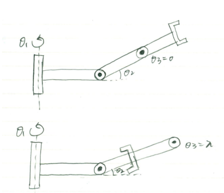

Figure 2: Elbow singularities

- Shoulder happens when $a_2c_2 + a_3c_{23} = 0$, as shown in Fig7.

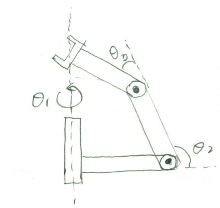

Figure 3: Shoulder singularities

Problem 03

(a) Since the first joint is revolute, the second is prismatic, and the third one is revolute:

$$z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad z_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad z_2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{1}$$

Since $l_1 = 2$, $l_2 = 1$, $l_3 = 2$:

$$J(p) = \begin{bmatrix} -(2+d_2)s_1 - c_1 + 2(-c_3s_1 + c_1s_3) & c_1 & 2(-s_3c_1 - c_3s_1) \\ -(2+d_2)c_1 - s_1 + 2(c_1c_3 - s_1s_3) & s_1 & 2(-s_1s_3 + c_1c_3) \\ 0 & 0 & 0 \end{bmatrix} \tag{2}$$

Thus,

$$J = \begin{bmatrix} -(2+d_2)s_1 - c_1 + 2(-c_3s_1 + c_1s_3) & c_1 & 2(-s_3c_1 - c_3s_1) \\ -(2+d_2)c_1 - s_1 + 2(c_1c_3 - s_1s_3) & s_1 & 2(-s_1s_3 + c_1c_3) \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \tag{3}$$

(b) The singularity occurs when:

$$det(J_{x,y,w_z}) = -d_2 - 2 = 0 \tag{4}$$

(c) Compute q using program (attached in matrix_calculator.py)

$$q = \begin{bmatrix} 0.44835819 \\ 0.83556903 \\ -0.19512195 \end{bmatrix} \tag{5}$$

The problem is underconstrained. The right pseudoinverse minimizes cost function $g(q') = \frac{1}{2}q' + wq'$

(d) According to $q' = q'^{*} + (I - J_r^+ J)q'_0)$, compute q' using program (attached in matrix_calculator.py)

$$q' = \begin{bmatrix} 0.44835819 \\ 0.83556903 \\ -0.19512195 \end{bmatrix} + \begin{bmatrix} 0.07317073 & -0.09580983 & -0.2421513 \\ -0.09580983 & 0.1254535 & 0.31707317 \\ -0.2421513 & 0.31707317 & 0.80137577 \end{bmatrix} q'_0 \tag{6}$$

(e) Compute using program (attached in matrix_calculator.py)

$$q = \begin{bmatrix} 1.42403811 \\ -1.4419873 \\ -3.42403811 \end{bmatrix} \tag{7}$$

The problem is overconstrained. The left pseudoinverse minimized cost function $g(q', v_d) = \frac{1}{2}(v_d - Jq')^T(v_d - Jq')$

(f) Compute using program (attached in matrix_calculator.py)

$$v_d = (-1, 2, 0, 0, 0, -2)^T \tag{8}$$

According to the result, z and w_x is unable to achieve.

Problem 04

(a) Given $q_i = 0$, $t_i = 0$, $q_f = 1$, $t_f = 2$, $q'_f = 1$, and $q''_c = 2$. Since the final velocity is not 0, we extend t_f to t'_f by $q'_f/q''_f = 0.5$ so that the velocity end at 0. Now new final position $q'f = 2.5$. According to LSPD equation, we can find that $t_c = 0.588562$. The number is calculated by a program, which is attached in file "calculator.py".

Then we can draw resultant position, velocity, and acceleration profiles as in Fig 4 5 6.

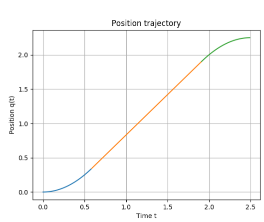

Figure 4: Position profile

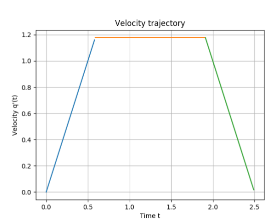

Figure 5: Velocity profile

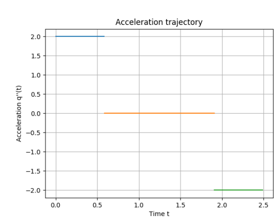

Figure 6: Acceleration profile

(b) Given $q_i = 0$, $t_i = 0$, $q_f = 1$, $t_f = 2$, $q'_f = 1$, and $q'_c = 1$. Since we don't know the acceleration, we cannot simplify this problem as the previous part.

However, if we look into velocity profile, we are able to represent final position q_f using t_c. As shown in Fig, which is the velocity profile, the area of shadow is final position, which is equals to $q_f = 2$. According to the graph we can know that velocity decrease to 0 when $t = t_f + \frac{2}{3}t_c$. Moveover, since $q'_c = 1.5$, $q''_c \times t_c = q'_c = 1.5$. Therefore:

$$\frac{1}{2} \times q''_c \times {t_c}^2 + (t_f - t_c) \times q'_c + (\frac{1}{2} \times q''_c \times {t_c}^2 - \frac{4}{9} \times \frac{1}{2} \times q''_c \times {t_c}^2) = 2$$

Finally we can get $t_c = 1.2, t'_f = 2.8, q'_f = 2.4$, and draw resultant position, velocity, and acceleration profiles as in Fig.

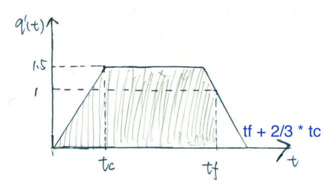

Figure 7: Shoulder singularities

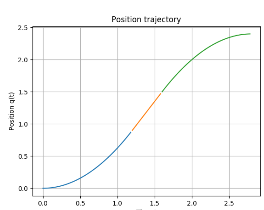

Figure 8: Position profile

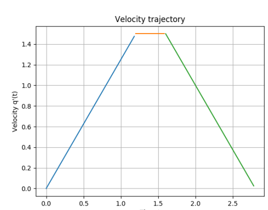

Figure 9: Velocity profile

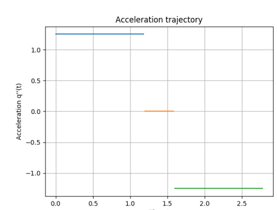

Figure 10: Acceleration profile

Problem 05

(a) The implementation is in file "p5.py". The computation of J is done by function computeJacobian(joints, DH, False), which takes joint variables and DH parameters as arguments. The last argument is used for writing output file, it doesn't have effect on calculation.

(b) The implementation of jacobian transpose algorithm is done by function updateJacoTranspose(e, joints, DH) in file "p5.py". Parameter e is initial error. The joint angles is recorded in file "jacobianTranspose_joint.txt" and the position of end effector after each movement is in file "jacobianTranspose_pos.txt".

To determine if the error is converge, I set a threshold so that the algorithm will stop if all elements is smaller than the threshold.

When set threshold to $1e-2$, number of iterations used to reach the desired pose is 103265. When set threshold to a smaller threshold $1e-3$, the number becomes 669725. Since the number is really large, I think it is better to draw trajectory of first 80 iterations. In the rest iteration, the arm just work in circle as the trajectory of first 80 iteration. The trajectory is smooth but goes into signularity.

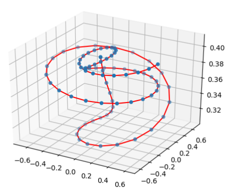

Figure 11: Jacobian Transpose

Sorry I didn't fix installation problem of ROS, therefore no ROS screenshot provided.

(c) The jacobian inverse algo is really quick. when the threshold is $1e-2$, number of iterations used to reach the desired pose is only 6. If the threshold set to $1e-3$ it will run 10 iteration. The DLS algo performs a little worse performance, but still fairly good, for threshold is $1e-2$ it terminates in 19 iterations. For threshold $= 1e-3$, it becomes 29.

The joint variables are recorded in files "jacobianInverse_joint.txt" and "DLS_joint.txt". The positions are recorded in files "jacobianInverse_pos.txt" and "DLS_joint.pos".

Trajectory for jacobian Inverse, it's not smooth, no singularity:

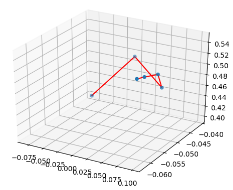

Figure 12: Jacobian Inverse

Trajectory for DLS, it's very smooth:

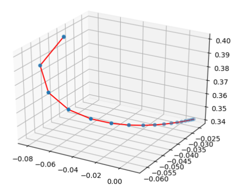

Figure 13: DLS