SECOND DIGIT OF WPI ID: 4

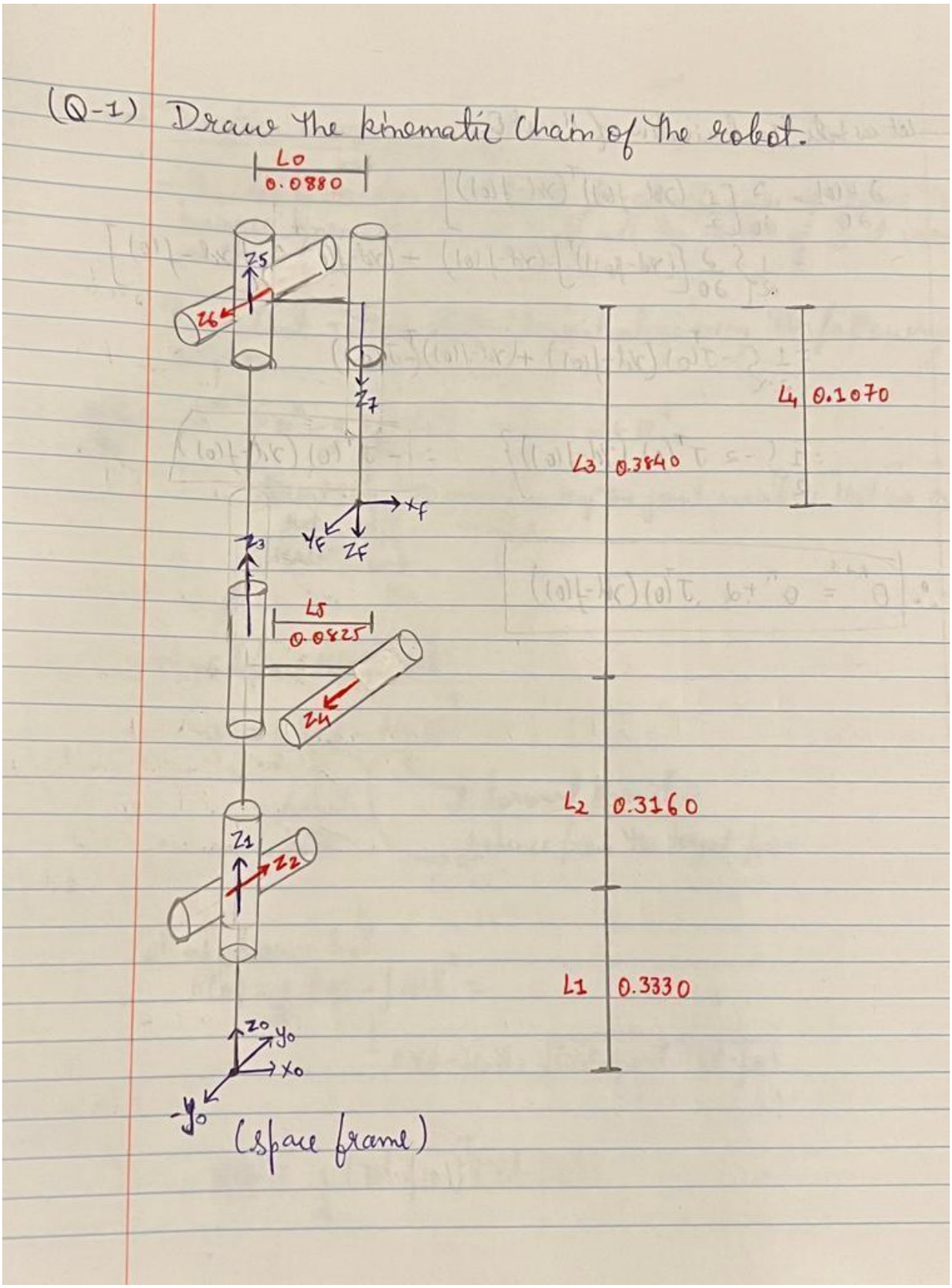
1.]

The following kinematic chain of the Franka Emika Panda is drawn and labelled as per the materials given in the Midterm Exam Paper.

Please note that the lengths L0, L1, L2, L3, L4 and L5 are corresponding to the links shown in the figure below.

Here,
L0 = 0.0880 m
L1 = 0.3330 m
L2 = 0.3160 m
L3 = 0.3840 m
L4 = 0.1070 m
L5 = 0.0825 m

(Q-1) Draw the kinematic chain of the robot.



2.] Calculation for screw axis of each joint with respect to the space is as follows:

(Q-2) Calculate each of the screw axes $\xi_i = (w_i, v_i)$ wrt the space frame

→ First we will calculate the home configuration →

$$M = \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$$

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \qquad P = \begin{bmatrix} L_0 \\ 0 \\ L_1+L_2+L_3-L_4 \end{bmatrix}$$

$$\therefore \boxed{M} = \begin{bmatrix} 1 & 0 & 0 & L_0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & (L_1+L_2+L_3-L_4) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

⟶ For screw axes S_i

→ $S_1 = [w_1 \; v_1]^T$

$w_1 = [0\;0\;1]^T$; $P_1 = [0\;0\;0]^T$; $v_1 = -[w_1]P_1 = [0\;0\;0]^T$

$\therefore \boxed{S_1} = [0\;0\;1\;0\;0\;0]^T$

→ $S_2 = [w_2 \; v_2]^T$

$w_2 = [0\;1\;0]^T$; $P_1 = [0\;0\;L_1]^T$

$$\therefore v_2 = -[w_2]P_2 = -\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ L_1 \end{bmatrix} = -\begin{bmatrix} L_1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -L_1 \\ 0 \\ 0 \end{bmatrix}$$

$\therefore \boxed{S_2} = [0\ 1\ 0\ -L_1\ 0\ 0]^T$

$\rightarrow S_3 = [w_3\ v_3]^T$

$w_3 = [0\ 0\ 1]^T$; $p_3 = [0\ 0\ 0]^T$; $v_3 = -[w_3]p_3 = [0\ 0\ 0]^T$

$\therefore \boxed{S_3} = [0\ 0\ 1\ 0\ 0\ 0]^T$

$\rightarrow S_4 = [w_4\ v_4]^T$

$w_4 = [0\ -1\ 0]^T$; $p_4 = [L_5\ 0\ (L_1+L_2)]^T$

$$v_4 = -[w_4]p_4 = -\begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} L_5 \\ 0 \\ (L_1+L_2) \end{bmatrix} = -\begin{bmatrix} -(L_1+L_2) \\ 0 \\ L_5 \end{bmatrix}$$

$$= \begin{bmatrix} (L_1+L_2) \\ 0 \\ -(L_5) \end{bmatrix}$$

$\therefore \boxed{S_4} = [0\ -1\ 0\ (L_1+L_2)\ 0\ -L_5]^T$

$\rightarrow S_5 = [w_5\ v_5]^T$

$w_5 = [0\ 0\ 1]^T$; $p_5 = [0\ 0\ 0]^T$; $v_5 = -[w_5]p_5 = [0\ 0\ 0]^T$

$\therefore \boxed{S_5} = [0\ 0\ 1\ 0\ 0\ 0]^T$

→ $S_6 = [w_6 \; v_6]^T$

$w_6 = [0 \; -1 \; 0]^T$; $P_6 = [0 \; 0 \; (L_1+L_2+L_3)]^T$

$$v_6 = -[w_6]P_6 = -\begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 0 \\ L_1+L_2+L_3 \end{bmatrix} = -\begin{bmatrix} -(L_1+L_2+L_3) \\ 0 \\ 0 \end{bmatrix}$$

$$v_6 = \begin{bmatrix} L_1+L_2+L_3 \\ 0 \\ 0 \end{bmatrix}^T$$

$\therefore \boxed{S_6} = [0 \; -1 \; 0 \; (L_1+L_2+L_3) \; 0 \; 0]^T$

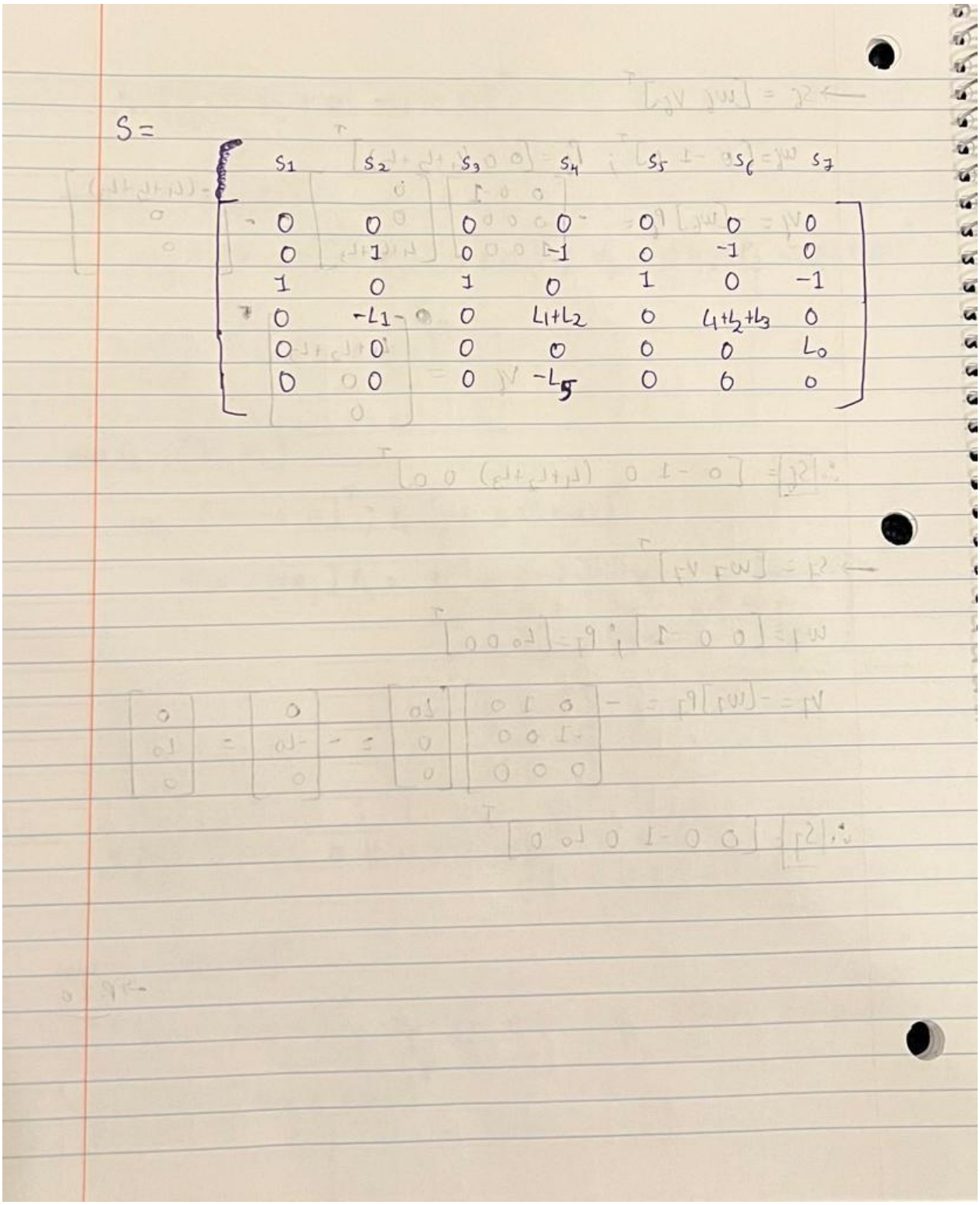
→ $S_7 = [w_7 \; v_7]^T$

$w_7 = [0 \; 0 \; -1]$; $P_7 = [L_0 \; 0 \; 0]^T$

$$v_7 = -[w_7]P_7 = -\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} L_0 \\ 0 \\ 0 \end{bmatrix} = -\begin{bmatrix} 0 \\ -L_0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ L_0 \\ 0 \end{bmatrix}$$

$\therefore \boxed{S_7} = [0 \; 0 \; -1 \; 0 \; L_0 \; 0]^T$

→ P.T.O



3.]

The forward kinematics of the given serial kinematic chain (Franka Emika Panda) is calculated in the poe.m matlab file.

S is a matrix that represents the collection of the screw axes of all the joints. It is a 6x7 matrix.

M is the homogenous transformation matrix that represents the home configuration of the given Franka Emika Panda robot.

I have also implemented 20 random tests to verify the correctness of the S and M matrix as well as the fkine function developed by me.

Please toggle plotOn variable to “true”, to see the robot in action.

4.]

The differential kinematics or the Space Jacobian of the given robot is calculated in the jacobian.m file.

Here,

S is a matrix that represents the collection of the screw axes of all the joints. It is a 6x7 matrix.

M is the homogenous transformation matrix that represents the home configuration of the given Franka Emika Panda robot.

This code calculates the Space Jacobian of only the home configuration (as per asked in the question.)

I have also written a simple script to verify the correctness of the obtained Space Jacobian.

5.]

The inverse Kinematics for Franks Emika Panda is implemented in ik.m matlab file.

Toggle the plotOn and ploterrorOn variable in poe.m file to “false” to run the code quickly as it will turn off all the plots.

However, to visualize the output please keep both the variables to “true.”

First, I implemented NEWTON-RAPHSON numerical method to find the inverse kinematic solution for the given robotic manipulator.

It iteratively calculates detlaQ using the following equation

- deltaQ = pinv(J)*(target_twist – current_twist) here,
    - deltaQ is a vector whose elements represent the change in joint values from the previous joint values.
    - pinv(J) is Moore-Penrose pseudo-inverse of the jacobian matrix. Pseudo-inverse is used to check abruptly high joint values.
    - target_twist is twist of the final desire pose of the end effector.
    - Current_twist is the twist of the current pose (at current iteration) of the end effector.

This method manages to converge to the solution and finds all the joint variables for all 36 points given in the question.

However, when we visualize the Error V/S Iterations graphs we see that it is not a truly converging graph for most of the points. Therefore, it manages to reach all the points, but it is not the best solution.

Here, we can see (in Fig.1) the Error V/S Iterations graph when the robot attempts to find a solution (joint variables) to reach the first target point (given in the question) form its initial position.

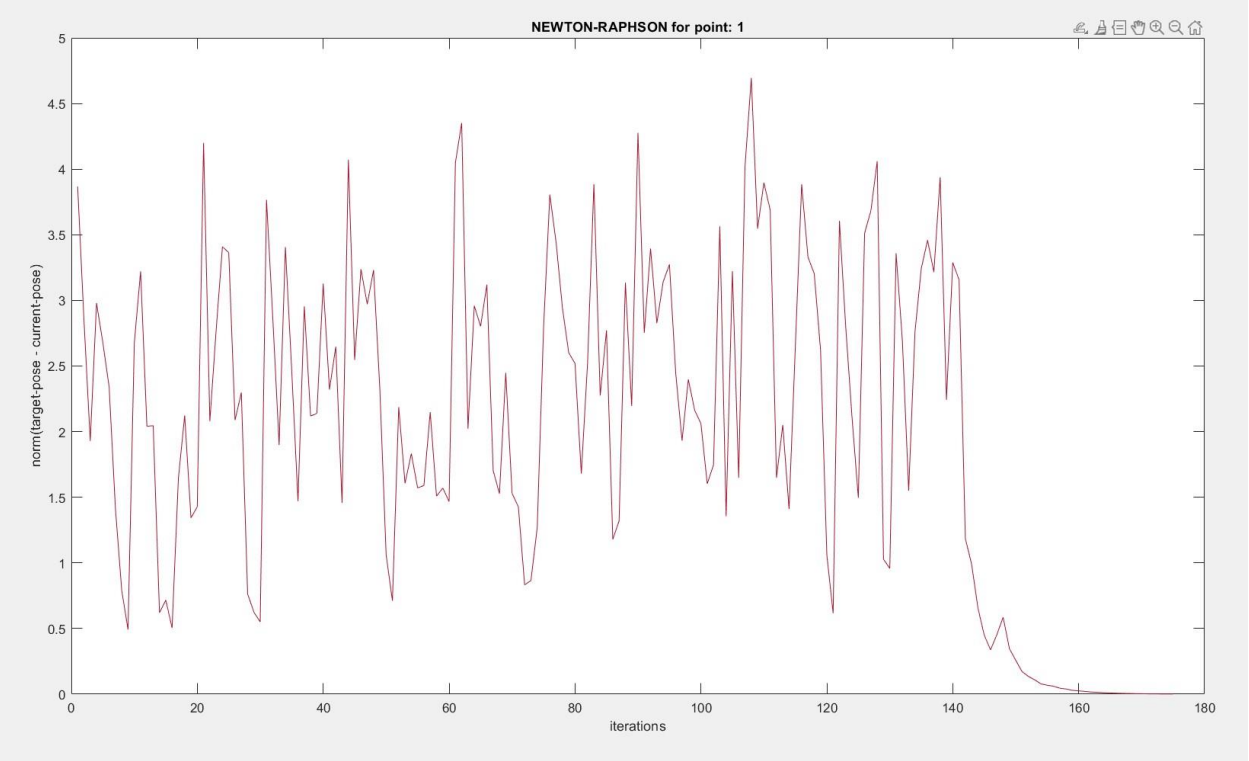


Fig. 1

Moreover, after observing the qList matrix, I saw the biggest failure of the Newton-Raphson method. As we can see in Fig.2 that the joint variables are very high. These joint variables are in radians and to implement a solution giving such high joint values is practically impossible.

And the time elapsed to calculate the solution for all 36 points was 6.590 s as seen in Fig. 3.

```
ans =

  Columns 1 through 4

        22.6890570712556        -303.861230836303         35.3725430380884        -130.422396843459
        22.6785458836897        -303.834859937849         35.3632443848489        -130.421468261771
        22.6360601472171         -303.76042435454         35.3371718809546        -130.424897662858
         22.551611830505        -303.645621350924         35.3043156206127         -130.44268988816
        22.4228263065246        -303.488952876267         35.2743529930013        -130.493307210746
        22.2598765202957        -303.306443340912         35.2632964827789        -130.603558473909
         22.080726950187         -303.11507431603         35.2705869789516        -130.774674222049
        21.9047681922022        -302.930879909881         35.2888605554893         -130.99524900196
        21.7461929774263         -302.75455433644         35.3081736365451        -131.245944262523
        21.6111210958974        -302.578054650255         35.3262351295137        -131.516228876341
        21.5059860482296        -302.407035571025          35.348612703994        -131.787403164081
       -352.610873724236        -93.1348451336918        -625.491814017566        -164.154194425541
         4956.5916965338        -5221.09309862637        -11704.8892710227         4937.78056114428
        4956.58632000577        -5221.21910834692         -11704.862146231         4937.60658137453
        4956.53681010217        -5221.37978383981        -11704.8232158748         4937.36528737274
        4956.49276266098        -5221.51670274447        -11704.7789328799         4937.15421759655
        4956.54697666895        -5221.58883072571        -11704.7989807404         4937.00074338989
        4956.60836262387         -5221.6093401893        -11704.8278845363         4936.92208000203
        4956.60836262387         -5221.6093401893        -11704.8278845363         4936.92208000203
        4956.50716331129        -5221.60140364555        -11704.7548058824         4937.00397637242
        4956.42294146483        -5221.55578567872        -11704.6645215634         4937.14165696375
        2987.93050222584        -4386.15229065294        -9532.07817778584         7766.31401830926
        2987.93595207768        -4386.26647540977        -9532.10748654731         7766.05955680528
        2987.98036774897        -4386.36538650342        -9532.15288441237         7765.88572857774
        2988.11406019712        -4386.38418229624        -9532.27979226942          7765.9372369487
        2988.32864219341        -4386.35710748001        -9532.43323458881         7766.17502056363
        2988.52235591682        -4386.33253109481        -9532.53795688107         7766.44624850475
        2988.66740918163        -4386.30602662225        -9532.62823280027         7766.71866823561
        2988.74633268667        -4386.28370541143        -9532.70242243425          7766.9667869681
        2988.75776042706        -4386.26468344239        -9532.74456835973         7767.19382464924
        2988.64410407218        -4386.24579064848        -9532.67182326744         7767.35861709717
        2988.41658002372         -4386.2318981026        -9532.48093971147         7767.46763820255
        2988.17968017416        -4386.26746330282        -9532.29995319375         7767.52322834049
```

Fig. 2

```
Progress: 100%
Test passed successfully.
----------------------Inverse Kinematics Test---------------------
Progress: 100%Elapsed time is 6.590593 seconds.
```

Fig.3

Therefore, to tackle this problem, I implemented the Damped-Least-Square method to find out the deltaQ and update it iteratively.

$J^* = \text{transpose}(J)*\text{pinv}(J*\text{transpose}(J) + (\text{lamda})^2*I)$ deltaQ

$= (J^*)*(\text{target\_twist} - \text{current\_twist})$

here,

- deltaQ is a vector whose elements represent the change in joint values from the previous joint values.
- pinv(k) is Moore-Penrose pseudo-inverse of the matrix k.
- target_twist is twist of the final desire pose of the end effector.
- Current_twist is the twist of the current pose (at current iteration) of the end effector.
- lambda is the damping factor.
- I is 6*6 Identity matrix

For this case I used lambda = 0.1

This method manages to converge to the solution and finds all the joint variables for all 36 points given in the question.

However, when we visualize the Error V/S Iterations graphs we see that it is not a truly converging graph for most of the points. However, the graph is much better than what it was in case of the Newton-Raphson method. Therefore, it manages to reach all the points, but it is not the best solution.

Here, we can see (in Fig.4) the Error V/S Iterations graph when the robot attempts to find a solution (joint variables) to reach the first target point (given in the question) form its initial position.

Moreover, after observing the qList matrix, I again observed the biggest failure of the Damped Least Square method. As we can see in Fig.5 that the joint variables are very high. Yes, these joint variables are very small as compared to the ones we obtained during the simple Newton-Raphson method. However, these joint variables are in radians and to implement a solution giving such high joint values is practically not reasonable. Many of the joints move more then 360 degrees, and such solutions are not practically feasible.

Whereas the time elapsed to calculate the solution for all 36 points was 0.17878 s as seen in Fig. 6. However, it is against our expectations that Damped Least square should converge slower, but as it does not overshoot a lot, it manages to converge faster than Newton-Raphson in this case.

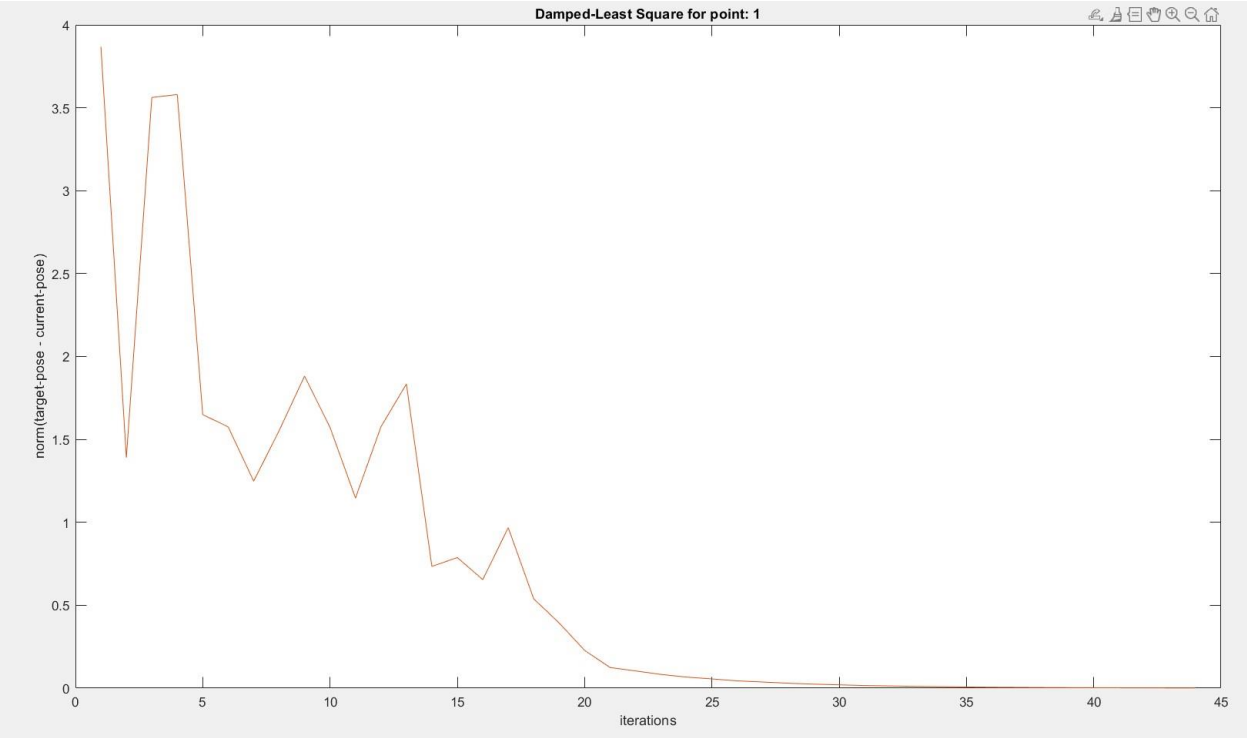

Damped-Least Square for point: 1
norm(target-pose - current-pose)
iterations
0
0.5
1
1.5
2
2.5
3
3.5
4
0
5
10
15
20
25
30
35
40
45

Fig. 4

Columns 1 through 4

-0.548166164293244	0.950847356417562	-8.35425108083738	-4.77965391876375
-0.552495033094146	0.915281505422636	-8.35383314939067	-4.7816483433976
-0.587530165067395	0.827255707756878	-8.35105552804293	-4.78857655013005
-0.665117937438964	0.694367366354783	-8.36118612020469	-4.80911008147467
-0.782250375543131	0.567588313359136	-8.39362263537083	-4.85704672420554
-0.921491886684676	0.508296014339571	-8.4066410773575	-4.9573692431621
-1.01199404232157	0.511296168947599	-8.42910509350489	-5.1206558444926
-1.01784714628377	0.549777048603033	-8.48397225414767	-5.3367087713941
-0.952864758098265	0.599211747132439	-8.55155229373747	-5.58925694839587
-0.831206705131247	0.646165924967637	-8.62259591667473	-5.85780008972137
-0.661176777338661	0.686999146838528	-8.70278504922406	-6.12623373801185
-0.463160579926	0.72256616679879	-8.79619049828747	-6.35983529495368
-0.3249438199139	0.703403578571626	-8.8804934911148	-6.41696017078257
-0.281117049364361	0.60133430145447	-8.93205998531766	-6.23663412827001
-0.265823430805435	0.487310292337351	-8.96972620807569	-5.98908850377134
-0.260112948287073	0.397884014190535	-8.99595751138311	-5.75961196099642
-0.263193186818998	0.341226832345126	-9.01270814152014	-5.58834902518384
-0.270004017251472	0.318693716837177	-9.02067384560484	-5.50151982570565
-0.270004017251472	0.318693716837177	-9.02067384560484	-5.50151982570565
-0.235611059744119	0.339980575085568	-9.02197253864055	-5.58770494385258
-0.142032977136124	0.391492423023353	-9.0301196125996	-5.75695210233605
0.0278667159438004	0.480855391466091	-9.05616230786265	-5.98965218986317
0.243788484730673	0.599310380584983	-9.11865911833651	-6.23677125887242
0.424582047441867	0.716148524509508	-9.2002621709625	-6.41785442173418
0.517926472258884	0.731577770385994	-9.27454236874628	-6.35861967500577
0.559037204670618	0.664903375471232	-9.34127570768295	-6.1286848668196
0.582635395683349	0.578929928103361	-9.40916134321752	-5.86024542843649
0.580909753981444	0.492719402161704	-9.48146185337438	-5.60118627276221
0.556815674812296	0.405713724579398	-9.56597750187165	-5.35622089749166
0.512810131679888	0.340215100795874	-9.67390681171968	-5.15385445033427
0.491660890175681	0.319380979699193	-9.84948512329738	-5.01770816800106
0.545504232449382	0.327213269209923	-10.0709753222319	-5.027422553973
0.159445129250646	0.291719206160718	-9.58529050986607	-4.98680005661656
0.0233490738925919	0.316564665491386	-9.437729783831	-4.91880461826963
-0.0140921539657609	0.350577990716344	-9.40674717231483	-4.88046125419672

Fig. 5

```
Test passed successfully.
----------------------Inverse Kinematics Test--------------------
Progress: 100%Elapsed time is 0.178782 seconds.
```

Fig. 6

Moving further, to optimize the solution even further I tried implementing the Gradient-Method to find out the deltaQ and update it iteratively.

deltaQ = alpha*transpose(J)*(target_twist – current_twist)

here,

- deltaQ is a vector whose elements represent the change in joint values from the previous joint values.
- target_twist is twist of the final desire pose of the end effector.
- Current_twist is the twist of the current pose (at current iteration) of the end effector.

- alpha is the learning constant.

For this case I used alpha = 0.35

This method was not able to converge to the solution and finds all the joint variables for all 36 points given in the question. But it was able to converge and find solutions for first few desired points given in the question.

However, when we visualize the Error V/S Iterations graphs we see that it is a truly converging graph for all the points for which it can find the solution. Therefore, it manages to reach few of the points, but it is the best solution for that particular configuration.

Here, we can see (in Fig.6) the Error V/S Iterations graph when the robot attempts to find a solution (joint variables) to reach the first target point (given in the question) form its initial position. And we can observe that it is a very smooth convergence.

Moreover, after observing the qList matrix, I again observed the biggest advantage of the Gradient method. As we can see in Fig.8 that the joint variables are reasonable. Therefore, these joint variables are in radians and to implement a solution giving such reasonable joint values is practically reasonable.

Whereas I am unable to comment on the total time elapsed as it did not converge for all the points.

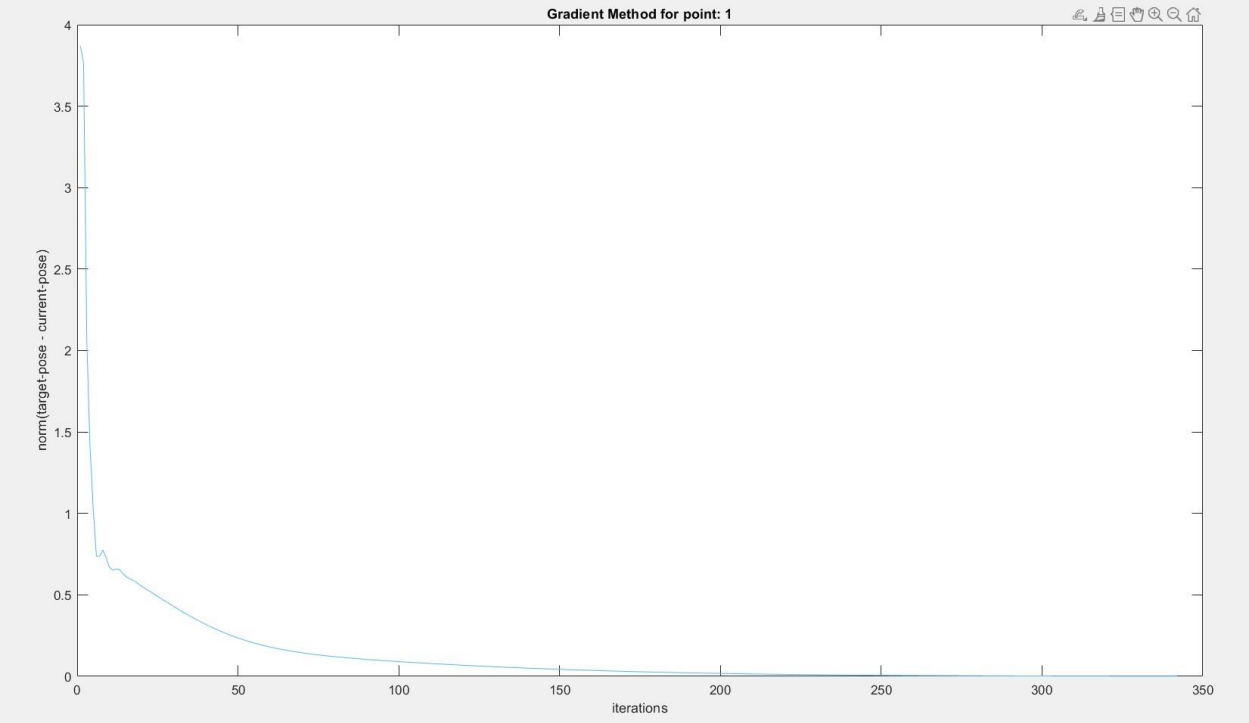


Fig. 7

```
ans =

  Columns 1 through 4

        1.16960934944976         1.38367474239514        -1.71905874384461        -2.41178728989231
        1.16682089614382         1.36868757777418        -1.74068288640849          -2.403685421613
        1.13679824822126         1.32304548441142        -1.79627615405875        -2.36820347983418
        1.05962650978294         1.24756955496195        -1.86384926077913        -2.29322379204263
       0.927000083693802         1.14283464275469        -1.91305403892273        -2.16015151847038
       0.740721725924633         1.01445041913747        -1.91431164543745        -1.95391238693053
       0.495268240282311        0.880358459208238        -1.84443741516615        -1.65625215651879
       0.146785655928584        0.779492533693235         -1.6887918767869        -1.21519239101201

  Columns 5 through 7

      -0.220299569624508        -1.89006877151224         -1.6150236260809
      -0.247961179825654        -1.89196667020898        -1.61003031354363
      -0.324448803557277        -1.88576528099095        -1.59623657594801
      -0.431226152433727        -1.86066273841715        -1.57565313737791
      -0.546669239425646        -1.82659751586991        -1.54703300226442
      -0.657105180244413         -1.8059960702364        -1.50118586262883
      -0.760231395143642        -1.82030297868158        -1.41224087884825
      -0.848799794107194        -1.87599449192516        -1.22306784920407
```

Fig. 8

From my observations I noticed that a method which combines the positives of both the Damped Least Square method and Gradient method will be best suited for our purpose.

Therefore, to get combined performance of both the Damped Least Square method and Gradient method, I implemented the following.

If error < 0.1

Damped Least Square method (Lambda = 0.001)

Else

Gradient method (alpha = 0.35)

Where, error = V2 norm of the difference of target_twist and current_twist.

This method manages to converge to the solution and finds all the joint variables for all 36 points given in the question.

When we visualize the Error V/S Iterations graphs we see that it is not a truly converging graph only for a few of the points, (here only 2). However, the graph is much better than what it was in case of the Newton-Raphson method and Damped Least Square method. It is comparable to the Gradient Method. Therefore, it manages to reach all the points, and it is the best possible solution for this problem.

Here, we can see (in Fig.9) the Error V/S Iterations graph when the robot attempts to find a solution (joint variables) to reach the first target point (given in the question) form its initial position.

However, we can see that this method failing to get a truly converging graph for the ninth point. We can observe that error overshoots as soon as it goes below 0.1 and while Damped Least Square method is being used. We can see this effect in Fig. 10.

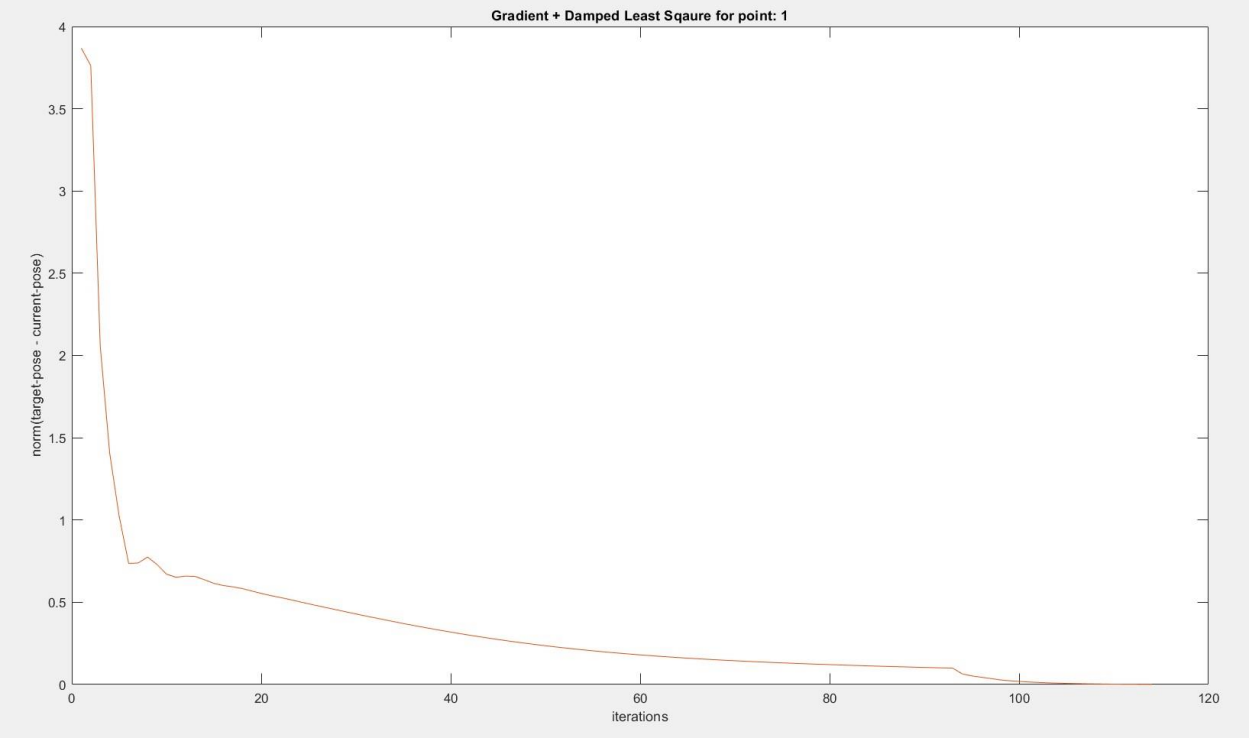


Fig. 9

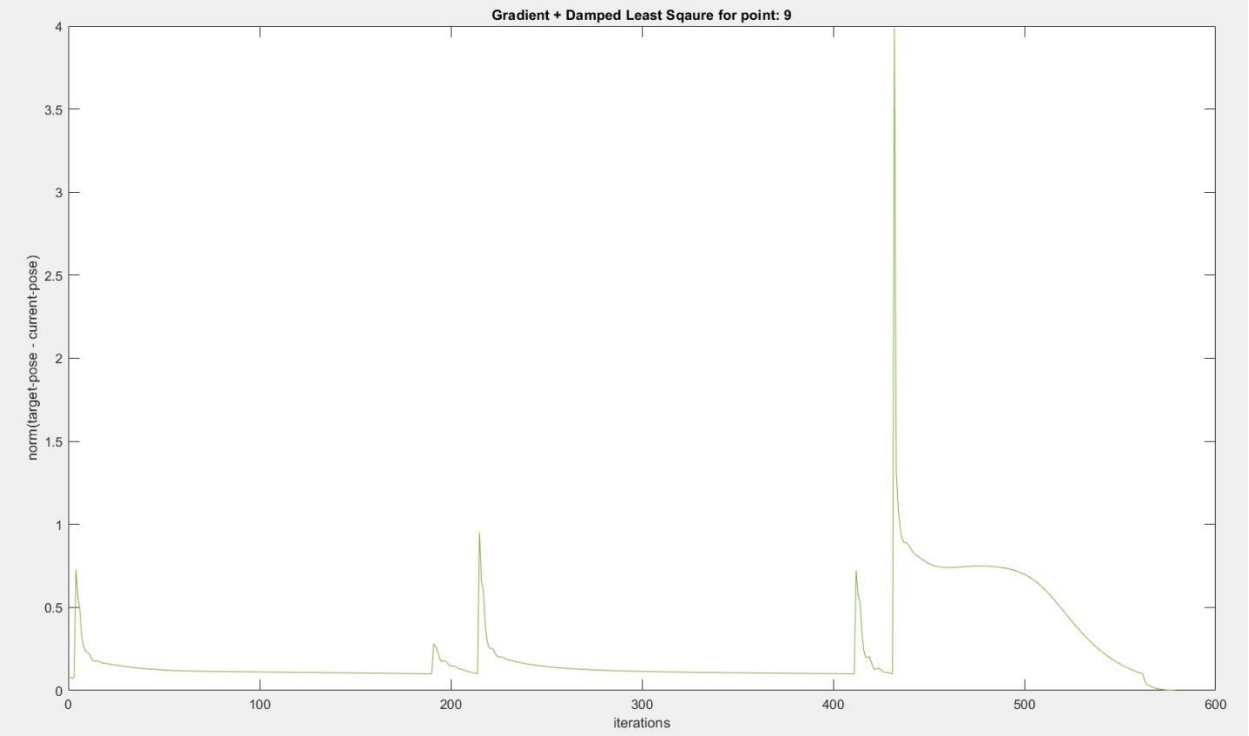


Fig. 10

Moreover, after observing the qList matrix, I witnessed the biggest advantage of this combined method. As we can see in Fig.11 that the joint variables are reasonable. It is not more than 360

degree for any joint configurations. Therefore, these joint variables are in radians and to implement a solution giving such reasonable joint values can be practically reasonable in many cases.

```
ans =

  Columns 1 through 4

        1.17161097699409         1.36743420006868        -1.71194918687788        -2.41161649472637
        1.16850145114079         1.35206510070866        -1.73361470132859        -2.40309379365292
        1.14032608756118         1.30479870226821        -1.78924556396524        -2.36965944614413
         1.0666394885008         1.22757775992506        -1.85450652130777        -2.29991170042356
        0.937028509126418          1.1168572693262         -1.9036678931135        -2.16526078464839
        0.75327124109138        0.981389896881318        -1.89900208838305        -1.95750832544022
        0.510748991008554        0.830708687700163        -1.80986426141293        -1.65964508992003
        0.157966092189432        0.695994979427487        -1.58266534531614        -1.23116505443067
         2.91306119202744        -1.74208712569882        0.393321012885915        -1.62802965485014
         2.84797958369155        -1.52281885993358        0.369449048781014        -1.35287605453543
         2.81640195469184        -1.31531167899899        0.334252723490352        -1.09011513042932
         2.65859586450491       -0.732773805955457         0.30703229215722       -0.0842694324064969
         2.77481957193655       -0.709788088833013        0.237191457654874       -0.126803016926889
         2.87261380864132       -0.595605042192697        0.172129269987624         0.05245467606819
         2.97038664675292       -0.471164285656521        0.117352587945042        0.301472799982145
         3.04730788952938       -0.381809736915742       0.0860207249088416         0.52074218542269
         3.08543110036037       -0.325658180188926       0.0724721005236789        0.692131376950765
         3.09484109085174       -0.300069768287111       0.0679069455346145        0.788201506907593
         3.09484109085174       -0.300069768287111       0.0679069455346145        0.788201506907593
         3.11298975913747       -0.326535900305753       0.0643051187386409        0.691132103602796
         3.16591185522761       -0.383689920895572       0.0556314028347803         0.51982766215242
         3.26658878673521       -0.476009619352278       0.0363702353226458        0.290805456276602
         3.40037893260321       -0.594407263050725      -0.00754514816635452        0.0541771299692056
         3.52686131119022       -0.723007608804931       -0.071422245445214       -0.145272281735192
         3.64064375425842       -0.725597373231368       -0.129937738757891      -0.0659271470896131
         3.73811246034698       -0.669597525548008       -0.205558538958433        0.152765278486864
         3.82436817793033       -0.590530130379322       -0.292331467120621        0.420175479618319
         3.88997793669114       -0.510197067972632       -0.393686477844302        0.689569044864346
         3.93453676971349       -0.440273844389039        -0.52104594236237        0.937887040432408
         3.96974406320878       -0.403784718297167       -0.698169674279249         1.14763206021015
         3.98155940122968       -0.435947175299352       -0.909626761089459         1.30313172817665
         3.94882348566978        -0.57712718687912        -1.12286280317187         1.39500970950024
```

Fig. 11

Whereas the time elapsed to calculate the solution for all 36 points was 0.494447 s as seen in Fig. 12.

```
----------------------Inverse Kinematics Test--------------------
Progress: 100%Elapsed time is 0.497747 seconds.
```

Fig. 12

Therefore, this method provides a fast and reliable solution for the concerned problem of finding the inverse kinematic solution for the Franka Emika Panda robot.

Finally, the robot is now able to trace the closed circular path, which can be seen in Fig. 13.

We can clearly see that the robot does not miss any of the target pose.

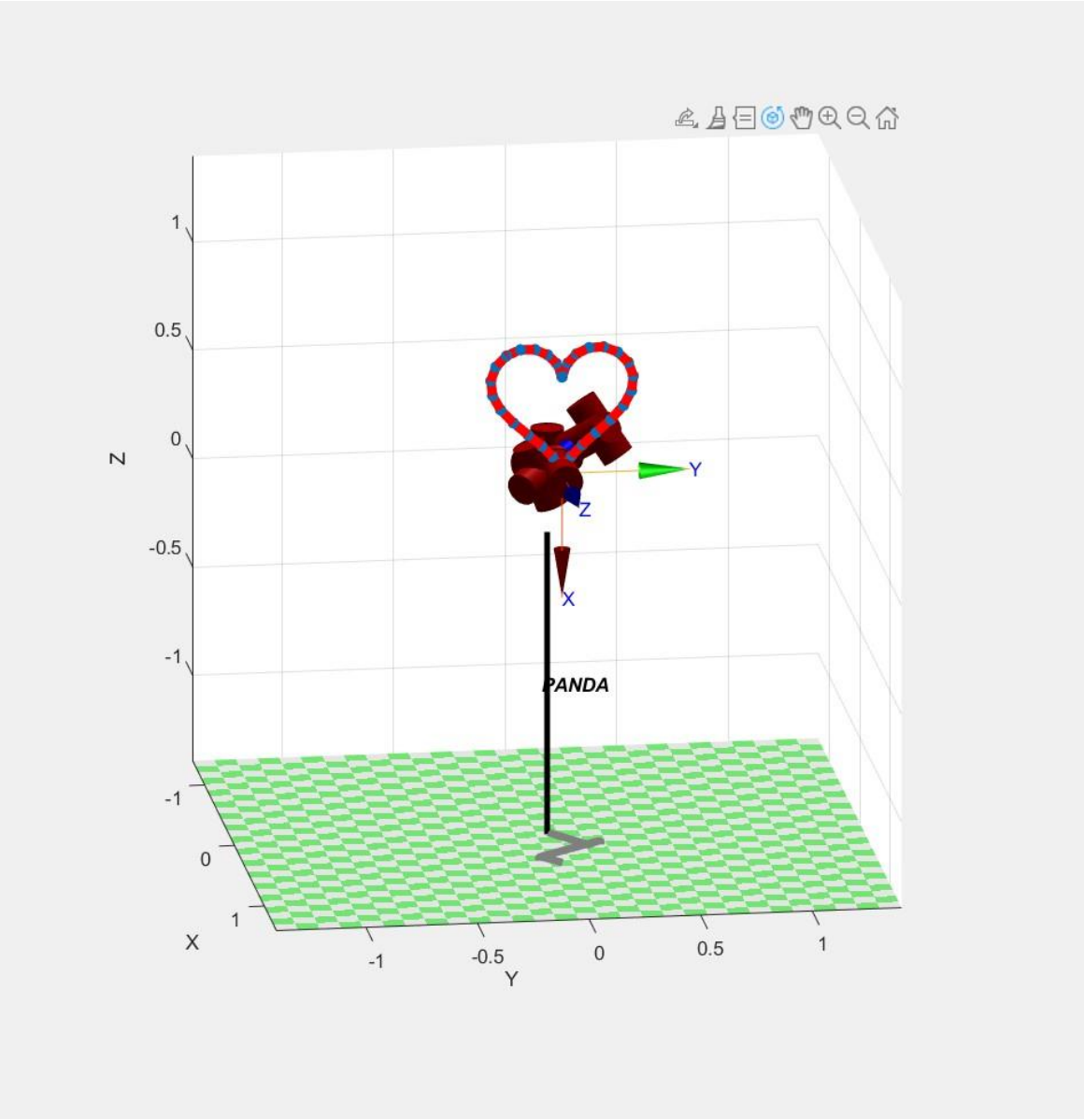


Fig.13