PROJECT MEDICAL ROBOTICS

MSc Biomedical Engineering 2017/2018

Supervisor: Evangeliou Nikos

Αδάλογλου Νικόλαος

Contents

1) Compute in symbolic form the matrix A_8^{12}

The Da Vinci Surgical System is a tele-manipulation system that consists of a surgeon's console, a patient cart with four interactive robotic arms, a high performance vision system and surgical instruments. In this report, we examine only one movable base with one arm. The kinematic diagram in the inertial coordinate system of the base and the arm shown in the Figure 1.

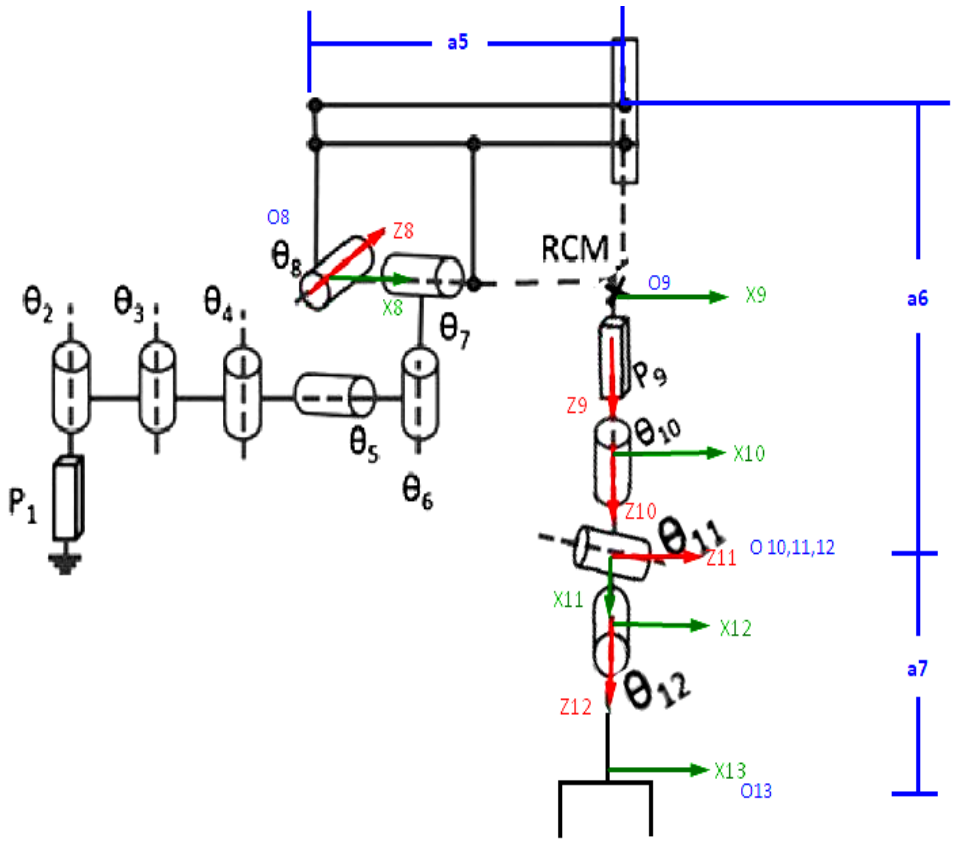


Figure 1: Kinematic diagram and Kinematic Configuration.

As we can see in the kinematic diagram in Figure 1, we have a 5-DOF manipulator. Using the kinematic configuration for each joint, we compute the Denavit – Hartenberg parameters as shown in the Table 1.

Parameter		Theta (°)	a (°)	r (mm)	d (mm)
Theta8	8-9	0*	-90	a5=300	0
d9	9-10	0	0	0	a6=200*
Theta10	10-11	-90*	+90	0	0
Theta11	11-12	+90*	-90	0	0
Theta12	12-13	0*	0	0	a7=18

Table 1: D-H parameters of the Robotic Arm

The asterisks in the Table 1, indicates those parameters that change during the motion of the joints.

After we calculate the parameters for each joint of the system, as shown in the Table 1, we can create a suitable code in Matlab using the symbolic toolbox, which will compute and create the transformation tables from one coordinate system to another and finally find the position and orientation of the grapple relatively to the inertial reference system (xo,yo,zo).

The general type of the transformation is the following:

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

Using the general transformation type, we create the A_8^9, A_9^{10}, A_{10}^{11}, A_{11}^{12} .

The total transformation of the coordinate system according to the inertial is given by the following relationship: $A_8^{12} = A_8^9 * A_9^{10} * A_{10}^{11} * A_{11}^{12}$. The results as given from the Matlab code shown in the Table 2.

A_8^{12}=

-cos(th12)*(sin(th8)*sin(th11) - cos(th8)*cos(th10)*cos(th11)) - cos(th8)*sin(th10)*sin(th12)	sin(th12)*(sin(th8)*sin(th11) - cos(th8)*cos(th10)*cos(th11)) - cos(th8)*cos(th12)*sin(th10)	- cos(th11)*sin(th8) - cos(th8)*cos(th10)*sin(th11)	300*cos(th8) - 18*cos(th11)*sin(th8) - d9*sin(th8) - 18*cos(th8)*cos(th10)*sin(th11)
cos(th12)*(cos(th8)*sin(th11) + cos(th10)*cos(th11)*sin(th8)) - sin(th8)*sin(th10)*sin(th12)	- sin(th12)*(cos(th8)*sin(th11) + cos(th10)*cos(th11)*sin(th8)) - cos(th12)*sin(th8)*sin(th10)	cos(th8)*cos(th11) - cos(th10)*sin(th8)*sin(th11)	300*sin(th8) + 18*cos(th8)*cos(th11) + d9*cos(th8) - 18*cos(th10)*sin(th8)*sin(th11)
- cos(th10)*sin(th12) - cos(th11)*cos(th12)*sin(th10)	cos(th11)*sin(th10)*sin(th12) - cos(th10)*cos(th12)	sin(th10)*sin(th11)	18*sin(th10)*sin(th11)
0	0	0	1

Table 2: **Total transformation of the coordinate system A_8^{12} Table**

2) Given the bounds of q compute points of the 3D-working space

Given the bounds:

$$\begin{bmatrix} 0^\circ \\ 0 \\ 0^\circ \\ 0^\circ \\ 0^\circ \end{bmatrix} \leq \overline{q} \leq \begin{bmatrix} 60^\circ \\ 200\text{mm} \\ 60^\circ \\ 45^\circ \\ 30^\circ \end{bmatrix}$$

We tested with 5 nested for loops the bounds to scan the 3d space. The step between the points was adjusted the computational speed. From the final matrix from the different combination of values we keep the first 3 rows of the 4th column of the matrix A_8^{12}as the 3d vector space.

It's important to mention that the given bounds of q were computed based on the initial state of the robot that the DH parameters were found. For example the revolute degree of freedom theta 10 has a range (0,60^0) and the initial condition is -90^0. So the computed range of the robot was (-90,-30).The same conclusion was applied in the other DOF.

Next, in Figure 2, 3, 4 we present a sparse and a dense view of the 3d space and also the 2d xz plot.

.

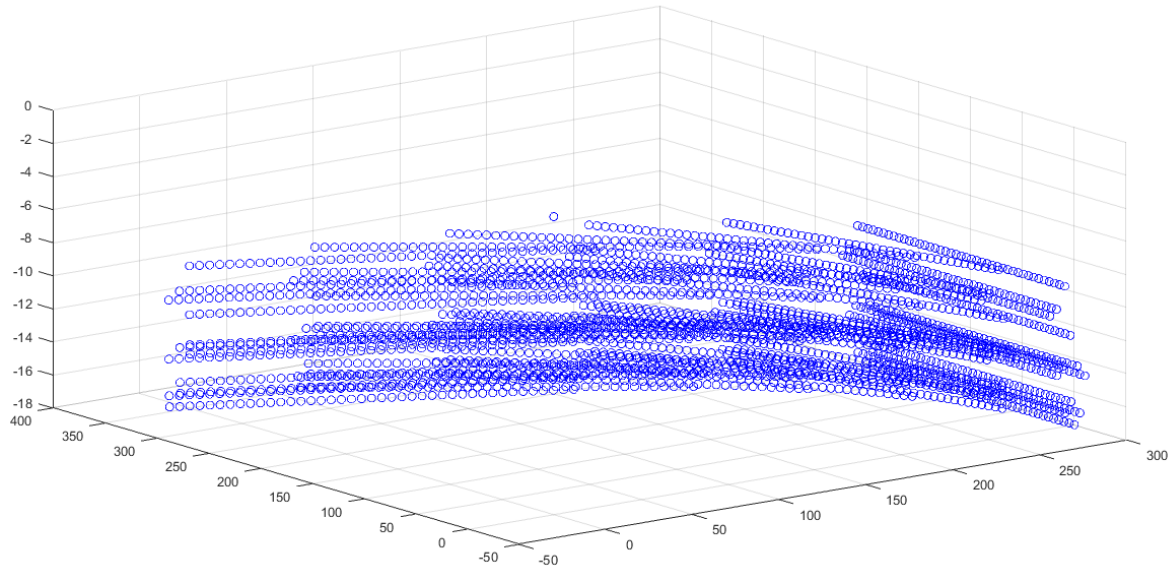


Figure 2: The 3-D working space of the robot (sparse view)

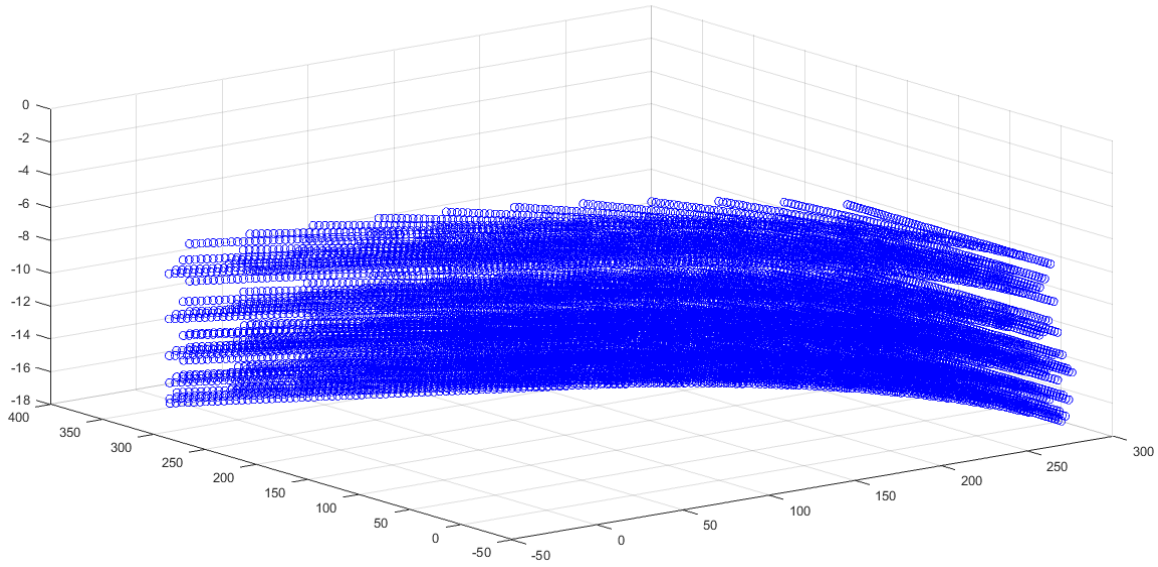


Figure 3: The 3-D working space of the robot (dense view)

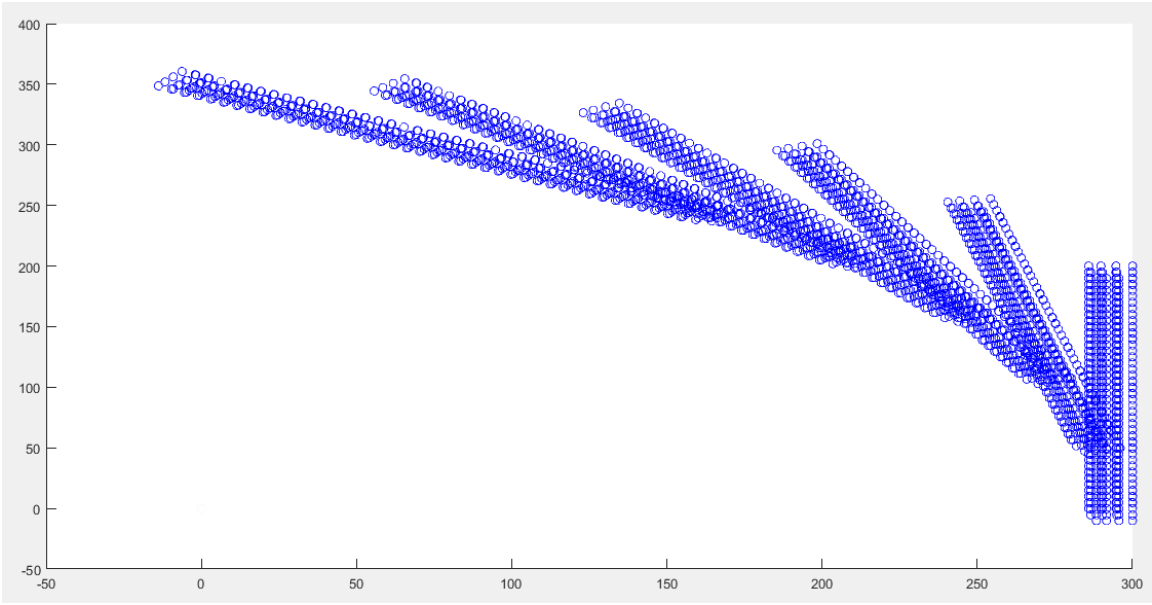


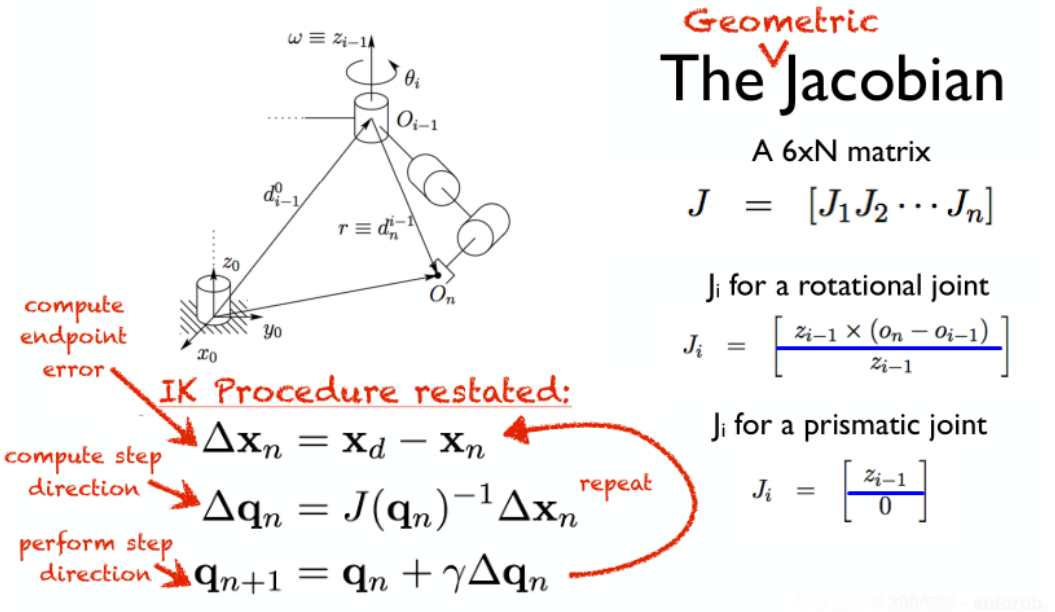
Figure 4: The 2-D working space of the robot in X,Y axes.

3) Solve the inverse kinematics problem, or given compatible values with the kinematic configuration of matrix T

We have studied three ways for the inverse kinematic problem. The first one is the geometric solution which mostly uses intuition. However, this method is quiet difficult to be applied for the Da Vinci problem. The second method is the analytical solution, which gives us a generalized solution for the inverse kinematic problem. Although, it is also difficult to compute analytically for 5 DOF. Furthermore, the matrix A_8^{12} is quite complex (does not have any element simple enough as cosine). The third method is iterative and gives a specific solution for given initial parameters and desired position.

The fundamental idea for the third method is gradient descent. It uses the information of the geometric linear velocity from the Jacobian matrix, which is similar to the first derivative of a single variable function. Actually the initial parameters are iteratively readjusted in the direction of the desired position, based on the inverse geometric linear Jacobian, which estimates linear velocity.

The summarization of the used method that was implemented is presented below:



$$\Delta \mathbf{x}_n = \mathbf{x}_d - \mathbf{x}_n \quad 3\text{x}1$$

$$\Delta \mathbf{q}_n = J(\mathbf{q}_n)^{-1} \Delta \mathbf{x}_n \quad 5\text{x}1 \quad 5\text{x}3 \quad 3\text{x}1$$

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \gamma \Delta \mathbf{q}_n \quad 5\text{x}1$$

In order to implement the IK problem we used:

1) An analytical distance function that we try to minimize between the desired and current position. We used the sum of the absolute differences between coordinates with threshold 5(mm).
2) A forward function that returns the position of the robot for the current parameters.(*Forward.m file*)
3) A function that calculates the Jacobian matrix for the current parameters.(*MyJacobian.m file*)

It is significant to mention that in order to compute the inverse of the Jacobian matrix, we calculated the pseudo-inverse so as the J is an orthonormal matrix.

The result of the *InverseKin.m* function that we have made gives the following results as shown below in the Figure 5.

```
>> InverseKin(   [ 1 180   -0.2   2.1   0]   , [33.354     335.1    -16.579] )
RESULTS:                    initial parameters                    desired position
-----------------------------------------------
Initial Position Of Simplified Davinci Robot
Init =
       10.045       331.97      -3.0869
Final Position Of Simplified Davinci Robot
final_pos =
       34.629       336.56      -14.312
Desired Position Of 3D-Workspace
Desired =
       33.354        335.1      -16.579
Iterations
        9946
Final Error in mm
error =
       4.9979
-----------------------------------------------
ans =    final parameters of the 5 DOF robot
       553.86       177.78        -25712        -51344          1692
```

Figure 5: Results of the inverse kinematic problem.

Comments: We chose the Learn Rate to be 0.05, which worked quite well for most of our tests. Given that the threshold is 5(mm). If the initial conditions are relatively good, the average iterations are close to 10.000.

In order to validate the results, we also computed a brute force search of the 3D space.

At this point, we have to mention that the method of the inverse kinematic that described above, probably it will not work successfully for every vector of parameters, since it is a follow-up function. We created a function (*Check_Param.m*) that checks the results and after try this, we conclude that not all of the solutions satisfy the initial constraints of the robot (especially theta12). The problem of the Inverse Kinematic is a quite complex problem, especially for 5 DOF.

An example of the function (*Check_Param.m*) as shown in Matlab presented below in Figure 6.

```
Parameter 4 in Range
>> Check_Param( [ 553.86       177.78       -25712       -51344         1692 ] )
Parameter 1 in Range
Parameter 2 in Range
Parameter 3 in Range
Parameter 4 in Range
Parameter 5 NOT in Range
>>
```

Figure 6: Indicative results of *Check_Param.m function*

It worth to mention that this solution is implemented both in the project file (*Final_Project.m → Q3),* and as a function in *(InverseKin.m).*

4) Select any spiral trajectory within your working space. Solve for 72 points the inverse kinematics problem by computing q.

The purpose of this question is to create a spiral trajectory that corresponds, with the best feasible way, to the already computed 3D working space of the robot, so we can solve the inverse kinematic problem.

The solution can be divided into three smaller solutions that are summarized below:

1. Determine the parameters of the spiral trajectory so as to be in the range of 3D working space.
2. Prune the total 3D working space, in order to pick the best possible matching points of the spiral trajectory, based on the geometry of the spiral.
3. Solve the inverse kinematic problem combining the inverse Jacobian approximation with the pruned search method.

4.1 Determine the parameters

To create the spiral trajectory, we use the equation below:

$$x(t)-x_c = r\cos\left(\frac{2\pi t}{T}\right), y(t)-y_c = r\sin\left(\frac{2\pi t}{T}\right), z(t)-z_c = d\left(\frac{2\pi t}{T}\right), r\neq 0, d\neq 0\,,$$

Where:

X_c	230
Y_C	200
Z_c	-14.5
d	0.75
T	1
r	20

Also, the range of the parametric equation (t) belongs to [-0.8, 1.3] where represents almost 2 cycles. The above values were selected after multiple trials to fit the spiral trajectory as best as it can in the 3D working space.

The spiral trajectory as it created in Matlab presented below in Figure 7.

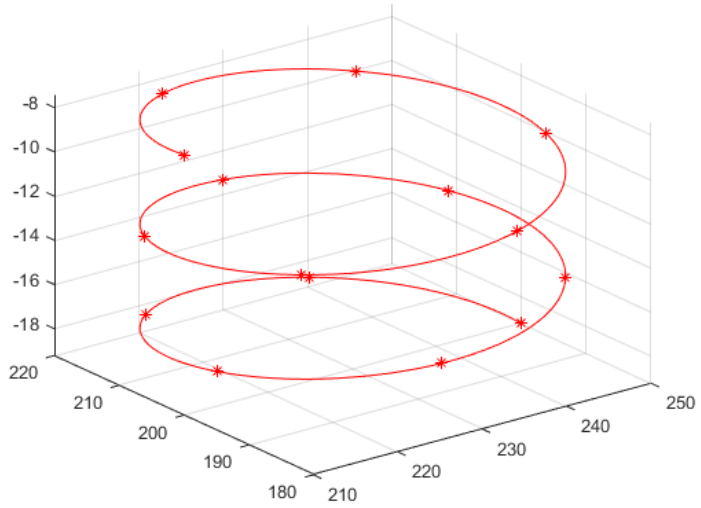


Figure 7 : Spiral trajectory in Matlab

The spiral trajectory in relation to the 3d working space of the robot shown in the Figure 8.

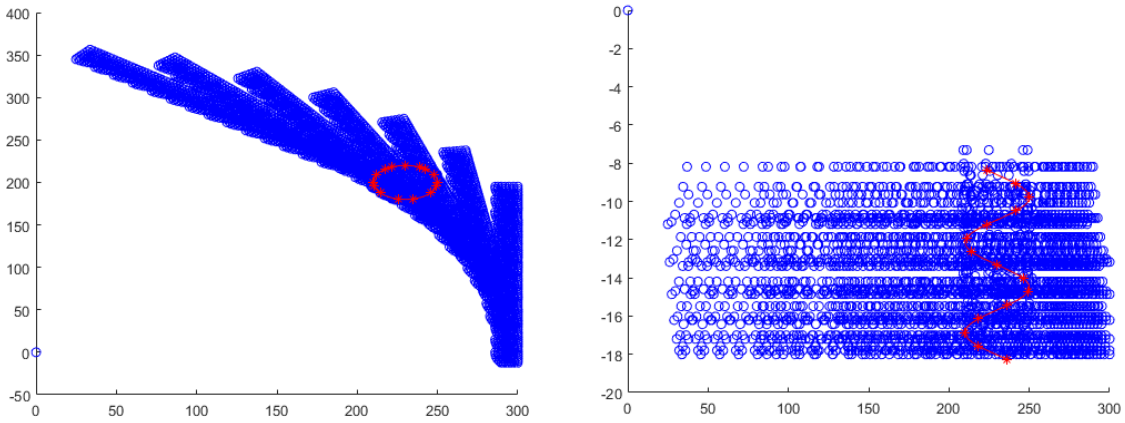


Figure 8: The spiral trajectory in relation to the 3D working space in X,Y axes and in a 3D plot

4.2 Prune the total 3D working space

The first approach was to match the n number of points of the 3D working space, with the spiral trajectory and then solve the inverse kinematic problem with the inverse Jacobian approach that was used in the previous question. After some experimentation, we faced 2 difficult problems. First it was computationally complex to compute-match the points with the predefined orbit (O(N*N)) and the final result of the matching was not perfect. From the XY and the XZ plot we observed that the orbit has specific ranges for X Y and Z (Table 2) and we matched only the points of the 3D working space that matched the boundaries. The second problem was that, because the points were not perfectly represented the trajectory, the inverse Jacobian kinematics approach did not converge in polynomial time.

X	208	252
Y	175	225
Z	-18.5	-7

Table 3: Ranges in X, Y, Z axes that used for pruning the working space.

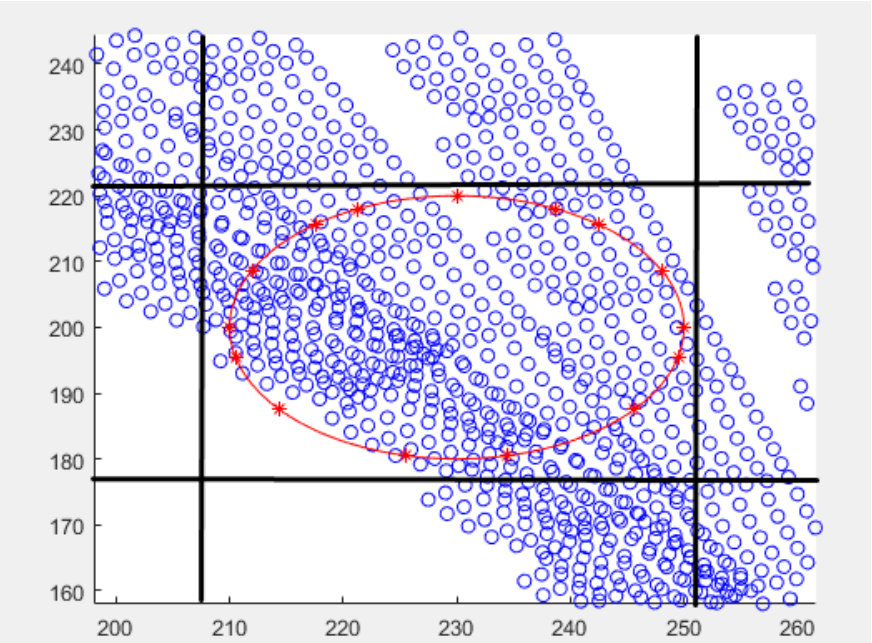


Figure 9: X, Y plot of the

We observed from the X, Y plot, that the desired points follow the equation of the cycle with center xc, yc and radius r. So we can prune the 3d space so that we can choose the X, Y points that belong to a circle with the same centre and a little bigger radius and do not belong to the cycle with the same center and a little bit smaller radius.

Summarizing:

$$Choose\ 3d\ points\ (x, y, z)\ that$$
$$(x - xc)^2 + (y - yc)^2 < (r + 2)$$
$$and\ (x - xc)^2 + (y - yc)^2 > (r - 2)$$

We choose the extra radius to be (**extra parameter q**)2 so for a circle with radius 20 that we choose the other two circles that we want them to be between have radius 18 and 22.

Adjusting the bigger and smaller circle with parameter 2 we observed that for too small values of q (for example 0.8) we have perfect solution to the X, Y level but very bad in the X, Z plane. So it's a tradeoff between choosing to little points that fit in the X, Y plane perfectly and it is computationally fast and taking more points from the whole 3d space with bigger computational complexity with a bigger probability to fit the workspace in the X, Z and Y, Z plane of the desired orbit.

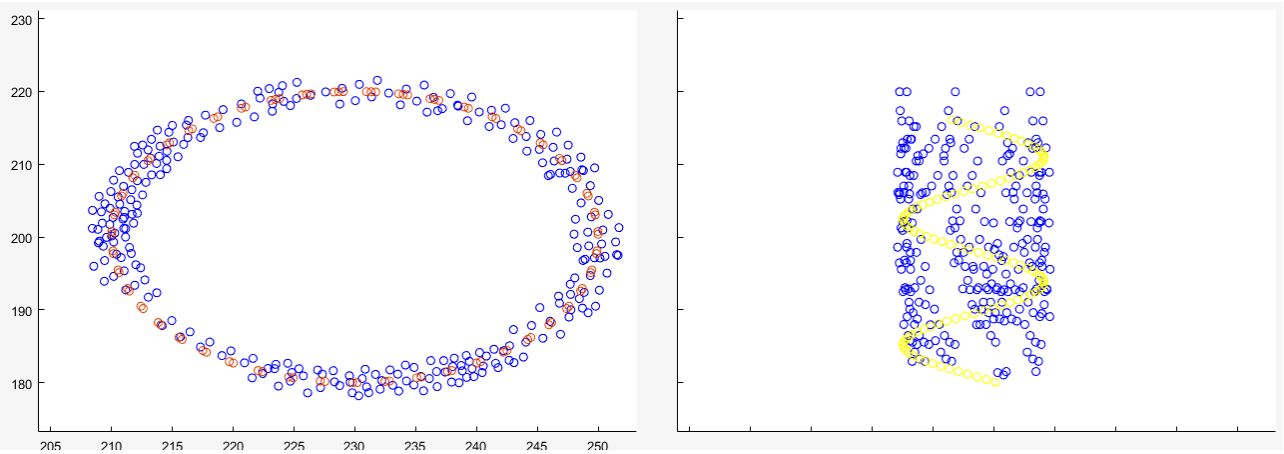


Figure 10: xy and xz plot after the selection of points

Finally, from the whole working space, using 0.12 for the ''for'' loops in Matlab, we reached to 1350 points out of the total space. The points in X, Y axis shown in Figure 10 above.

4.3 Solve the inverse kinematic problem

We already concluded that, the Jacobian function is not always converging, especially when the distance between the initial position and the final position is big enough. The final approach to the inverse kinematic problem is to choose the N points of the spiral trajectory as desired position. Next, we test the inverse Jacobian method for 20.000 iterations and accept its solution only if the sum of absolute error between the desired and the final position is less than 5(mm). Another reason that we chose as distance function the sum of absolute error is that it gives a direct interpretation and is in the same units (mm) with the movement of the robot. In case the Jacobian inverse kinematic method does not give an acceptable solution we perform a pruned search to find the best match of the spiral trajectory to the 1350 points that were found in question 4.2. Choosing more points to describe the spiral trajectory in the specific range [-0.8, 1.3], the inverse kinematic Jacobian approximation performs better and converges more quickly. Although, the problem of convergence exists again. We observed after plotting the x, y plane that the orbit diverges significantly from the desired of the spiral trajectory.

This is the reason that in the last implementation we used only the pruned search with one extra condition. The problem lies in the Z-axis. Therefore, the extra criterion was to take into account the absolute difference between the desired position and the 3D working space to be bellow a specific threshold. The units in Z-axis are not the same with the other two (x, y) but are one order of magnitude less we create another threshold so as to minimize the distance error in Z-axis. The extra constriction is the absolute distance between the axis to be bellow 0.4.

The pruned search always converges to a solution since it only checks the points of interest. It performs quite well and the values of the parameters of this solution are always valid, because they are points of the 3D working space. For 148 points the average error in mm was 4.06 with maximum value of 9.8.

Figure 11 and Figure 12 represents with red circles the points of the trajectory and with blue circles the inverse kinematic solution of the robot for 148 points. The first one in X, Z axis and the second one X, Y axis.

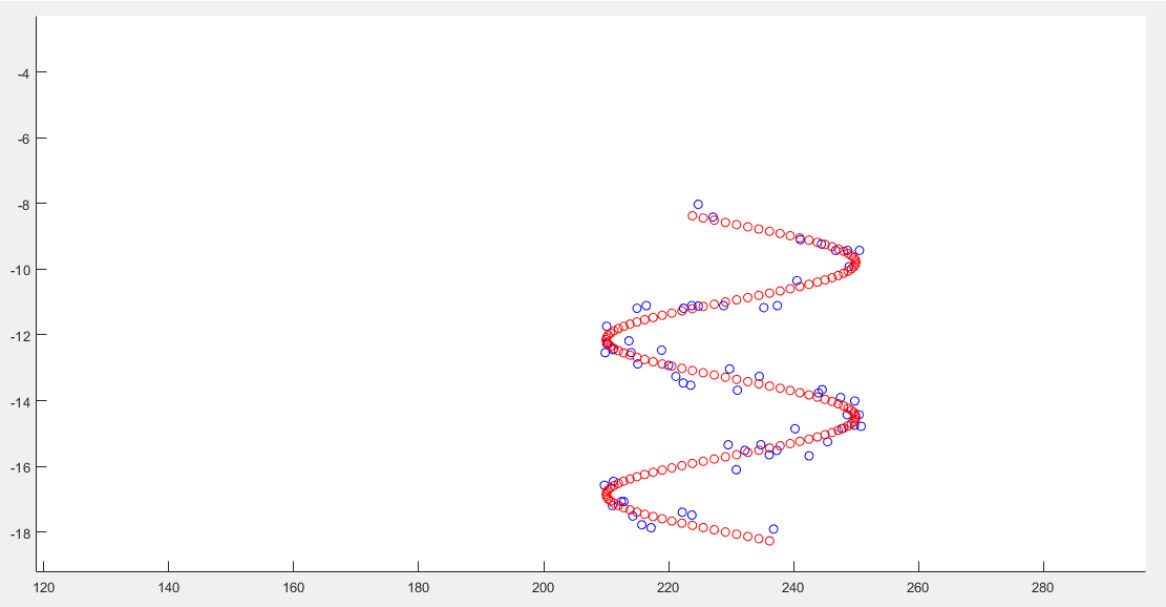


Figure 11: xz plot of the final approximation(blue) of the predefined orbit(red) for 148 points

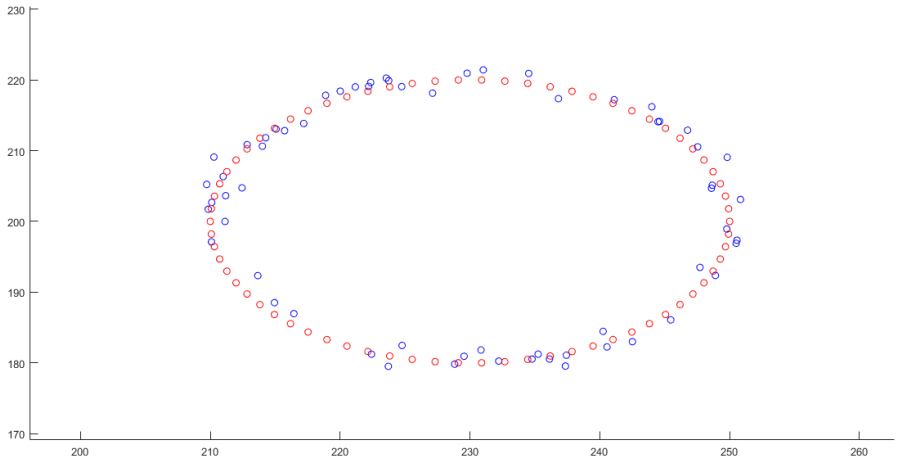


Figure 12: xz plot of the final approximation(blue) of the predefined orbit(red) for 148 points

5) Under the assumption that two successive points are connected by a third order polynomial with a stop-and-go trajectory plot the joint trajectories and the resulting end-effector trajectory

We found the 148 points of the orbit and their parameters to achieve the desired positions from Q4 we have stored them in memory (*file Q4_answer_data_final.mat)*. The next step was to approximate, successive points the 3rd order polynomial. We created a Matlab function (*Third_order_solver.m*) that calculated the values of the polynomial with velocities both at start and end point equal to zero. The following plot gives us an example of the function `Third_order_solver( t_start=0 , t_end=10 , start_point=30 , end_point=60)`

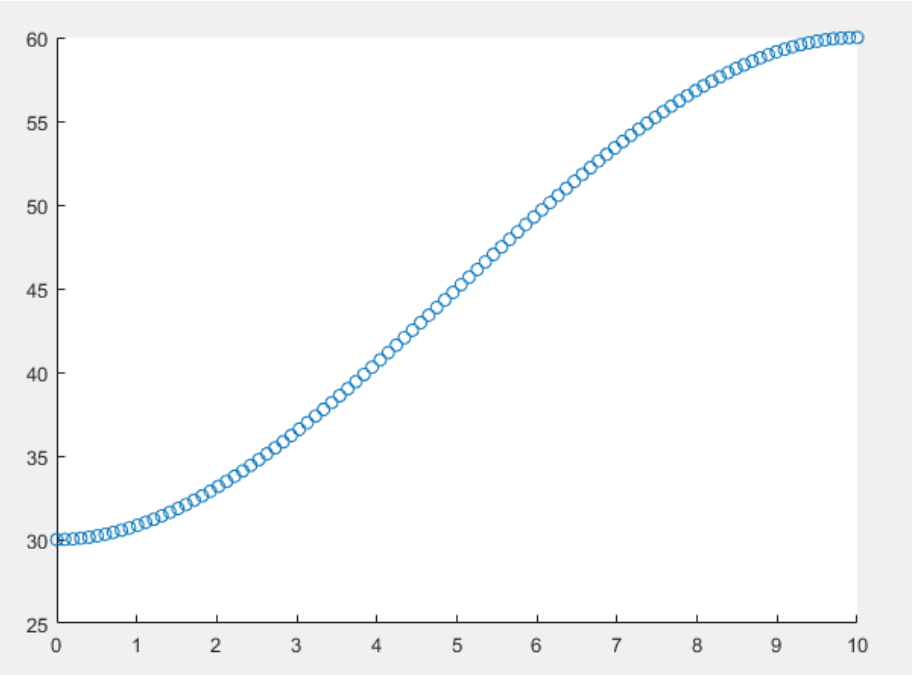


Figure 13: Third order polynomial

The math behind the 3rd order polynomial to find the coefficients in matrix form is:

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \bullet \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ \dot{q}_{0=0} \\ q_f \\ \dot{q}_{f=0} \end{bmatrix}$$

Table 4: 3rd order equation

The calculation of the inverse matrix is usually feasible. In any case, we again use pinv(matrix) instead of inv(matrix). Now we must apply this for every consecutive pair point and for every DOF. In the Figures 14, 15, 16, 17, 18, we plot the result of the 5 parameters (th8, d9, th10, th11, th12) of our DOF for all the movement of the robot.

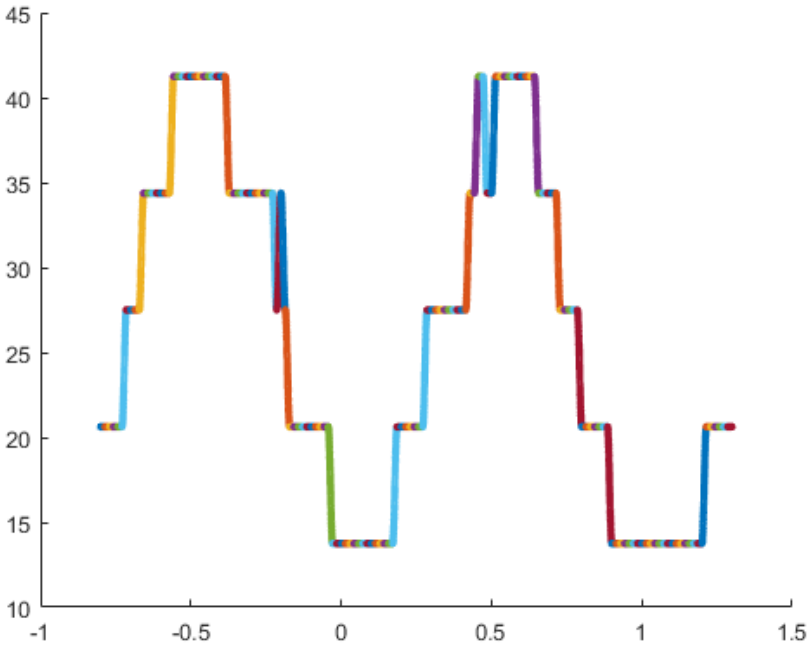


Figure 14: Change of th8 (°)

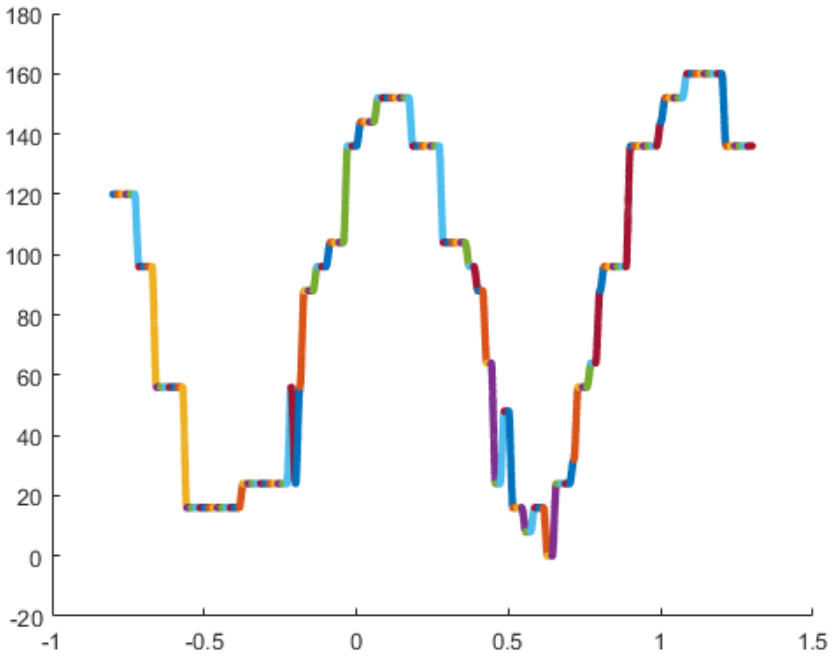


Figure 15: Change of d9 (mm)

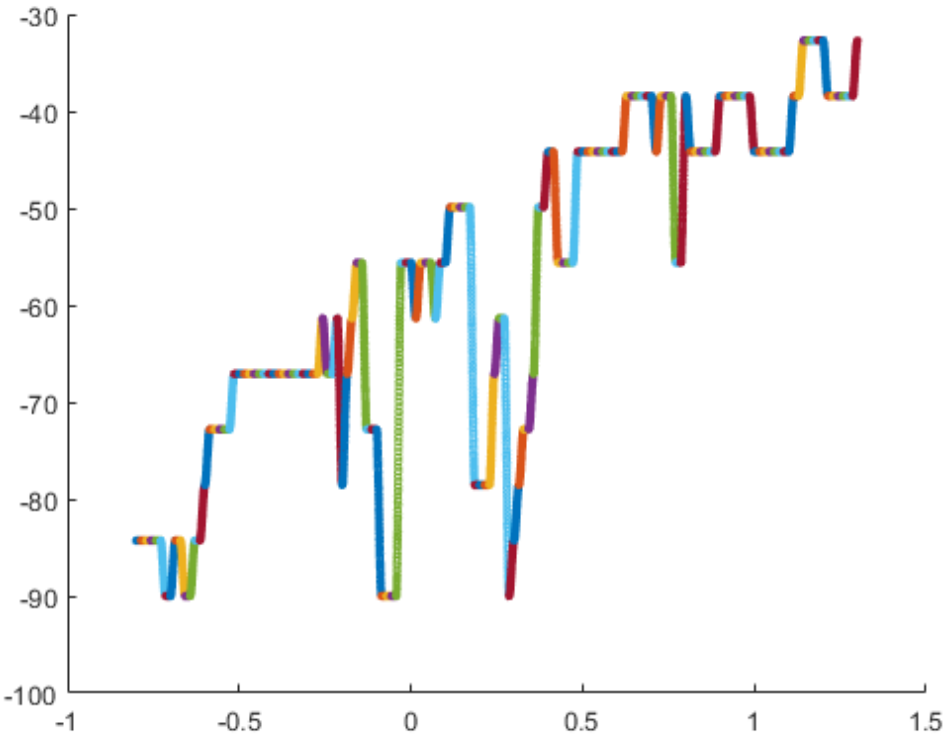


Figure 16: Change of th10 (°)

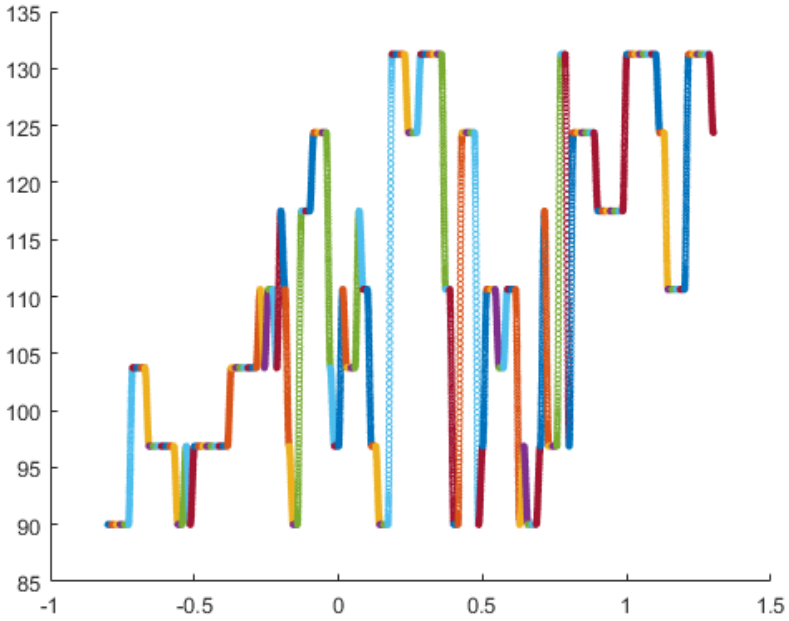


Figure 17: Change of th11 (°)

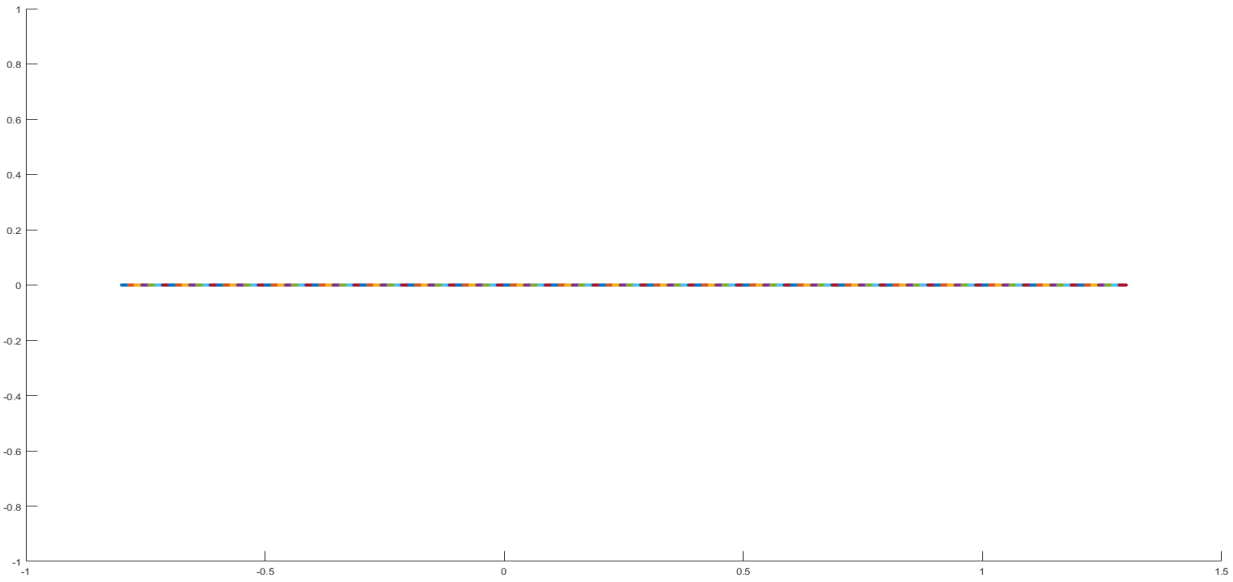


Figure 18: Change of th12 (°)

We observe that the last DOF th12 does not change because it does not change the 3d position of the robot. Furthermore, we validate that the degrees are inside the desired ranges that are represented in degrees for angles (for visualization purposes) and in mm for the prismatic DOF.