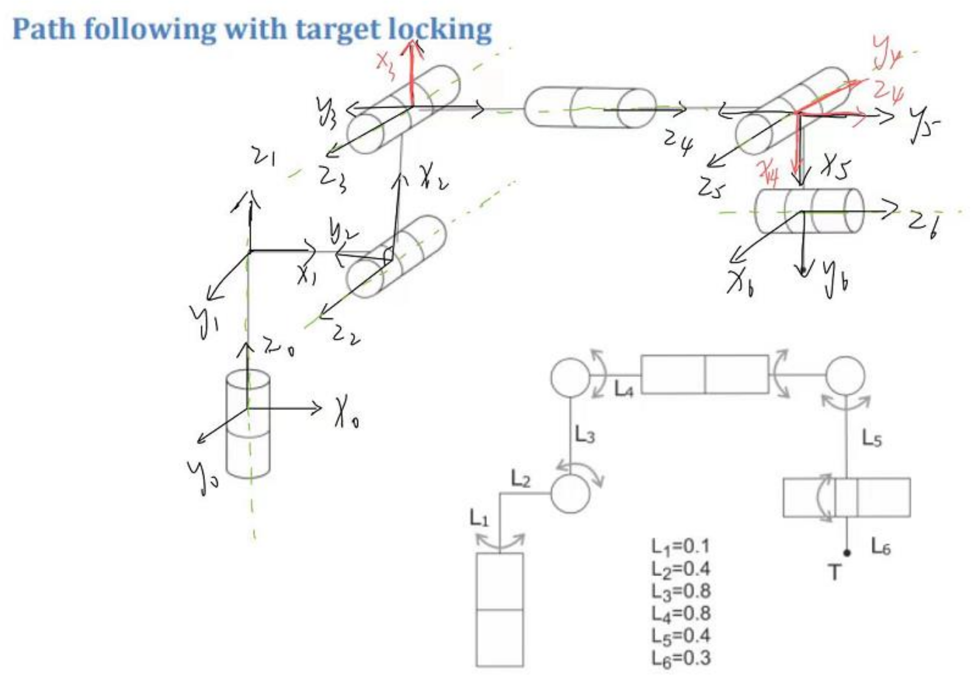
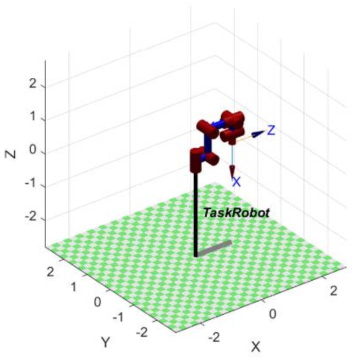


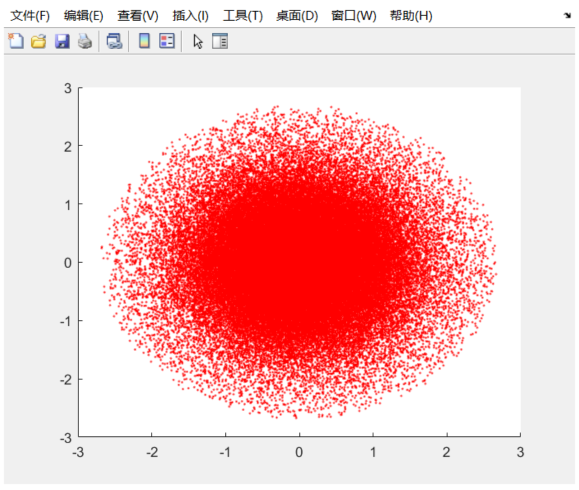
DH 表	a_{i-1}	α_{i-1}	d_i	θ_i
1	0	0	L1	$\theta 1$
2	L2	Pi/2	0	Pi/2 $\theta 2$
3	L3	0	0	0 $\theta 3$
4	0	pi/2	L4	Pi $\theta 4$
5	0	Pi/2	0	$\theta 5$
6	L5	-pi/2	0	$\theta 6$

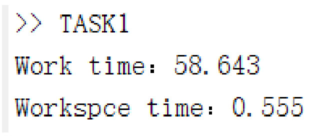
Task1:

The correct DH table is created from the given diagram. Each joint of the robot is connected by a link and each link has a coordinate frame. Z-axis corresponds to the rotation axis of the joint, the X-axis is the direction pointing from one joint to the next and is perpendicular to the common plumb line and the Y-axis is defined according to the right hand rule. The pose of each joint is described with reference to the previous frame.



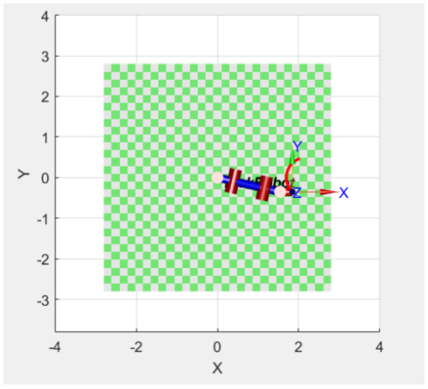
The problem I encountered in building the DH table was that the off-set had to change when the X3 was pointing in a different direction, so it was important to pay special attention to where the X-axis was pointing, as this would affect the following metrics.
Based on this DH table, and using the modified DH convention, the correct robot structure is shown.
The pos_static_trajectory file is then imported so that the robot moves the end-effector following a liner trajectory along a sphere, giving the motion process shown in the attached Task1.mp4.
To plot the workspace you need to set the limits of each joint. Joints 1-5 all have a range of -180 to 180 degrees and joint 6 has a range of 0 to 360. 100,000 points are taken to plot the workspace of this robot and as the sixth joint has no effect on the workspace it is set to 0.

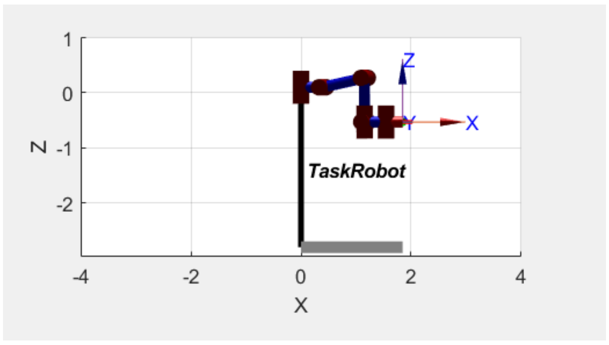




```
>> TASK1
Work time: 58.643
Workspce time: 0.555
```

As can be seen from the graph, Task1 has almost no error.





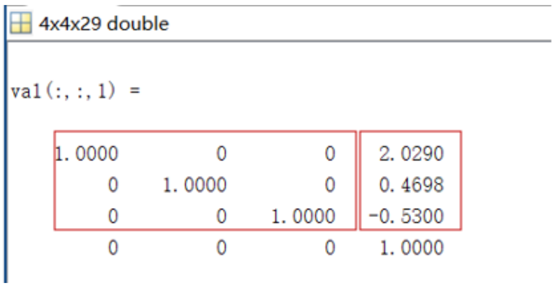
Task2:

The difference between Task2 and Task1 is to ensure that the end-effector is always perpendicular to the surface of the ball, so the joint must move with it in order to remain perpendicular to the surface.

The given pos_static_trajectory is transposed to a 29X3 T-matrix and then the flush transformation (4×4) is performed by T4 = Transel(T), (4×4× M) is a sequence of flush transformations and the rows of (M×3) are the translational components of the corresponding transformations in the sequence such that (:, :, i) corresponds to the i-th row of T.

Use angvec2r Convert angle and vector orientation to a rotation matrix.

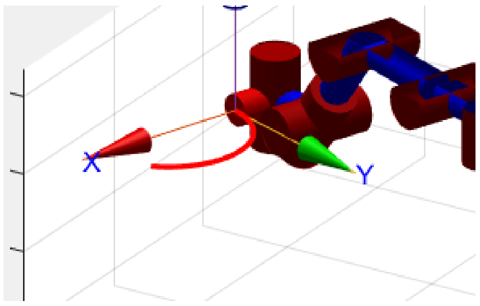
Use rt2tr to Convert rotation and translation to homogeneous transform.


4x4x29 double

```
val(:, :, 1) =

    1.0000         0         0    2.0290
         0    1.0000         0    0.4698
         0         0    1.0000   -0.5300
         0         0         0    1.0000
```

The trajectory of Task2 is shown in TASK2.mp4. The actual trajectory of Task2 is shown to coincide almost exactly with the ideal trajectory, and the error is very small.



Task3:

In Task3 we first repeat Task1, setting the end-effector to a constant speed of 0.1m/s

First importing the points of the pos_beat_trajectory, we have to inverse the angular velocity of each joint by the linear velocity and can find the relationship between the end linear velocity and the angular velocity of each joint.

An initial position is given first, then equations are made to calculate the required linear velocity and time. For example V=Jacob*W(Articular angular velocity) can then be inverted toW=V*(inverse_J).

This motion will only involve the linear velocity of the X and Y axes, as the robot is not moving in the Z direction. The symbolic variables X-axis linear velocity, Y-axis linear velocity and time are created via syms. By using the terminal linear velocity, the distance on the X and Y axes is calculated cyclically for each segment, and the linear velocity of each segment can be obtained by dividing the distance by the time. This equation is solved using the solve function to obtain V. The Jacobian matrix is calculated by the Jacob0 method to map the joint velocities

to the spatial velocities of the end-effectors in the world coordinate system, which are found by pinv pseudo-inversion to the joint velocities, and then superimposed to calculate the values of all the angular velocities required.
This motion trajectory is shown by T3a.MP4.

Repeating the second task, the trajectory is divided into 28 segments as there are 29 points, the total linear velocity V is known and the distance can be found in the X-axis direction, the linear velocity in the Y-axis direction and the time. The relationship between the end velocity and the joint angular velocity can be mapped by pseudo-inverse Jacob with the following formula: W=V*(inverse_J).
By knowing the angular velocity of the joint at the previous position, the time and the linear velocity, the angular velocity of this one joint can be found, and this can be used to find the angular velocity of all joints by looping. This motion involves the angular velocity in the Z-axis direction, as the robot rotates around the Z-axis, and from the 4X4 matrix we can co-cos the angle to get the angular velocity of the Z-axis, and combine these to get a matrix.
This motion trajectory is shown by T3b.MP4.
The final task can be seen through T3.mp4 that the actual trajectory and the ideal trajectory do not exactly coincide. To solve this problem, the first thing that can be done is to take more points and set more points out of 29 to make the trajectory coincide better.