MEAM 520

Lab 4: Velocity Kinematics

Xiaozhou Zhang / Xinlong Zheng

xzzhang@seas.upenn.edu

xinlongz@seas.upenn.edu

1 Method

1.1 Forward velocity kinematics

The task is asked to derive forward velocity kinematics. We manage to get the Jacobian matrix in the geometric way(using cross products). The flowchart is shown as Figure 1-1.

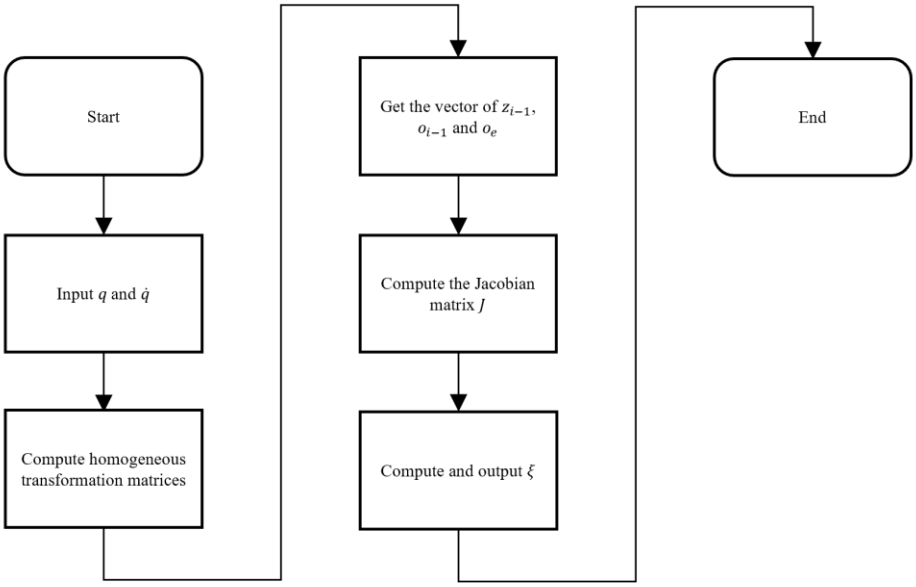

Figure 1-1 Flowchart of forward velocity kinematics

In our Lynx robot, for any given configuration q, as we have seen in class and textbook (SHV p133), the upper half of the Jacobian $J_v(q)$ is given as

$$J_v(q) = [J_{v_1}(q) \quad J_{v_2}(q) \quad J_{v_3}(q) \quad J_{v_4}(q) \quad J_{v_5}(q)]$$

in which the i^{th} column $J_{v_i}(q)$ is

$$J_{v_i}(q) = z_{i-1}(q) \times (o_e(q) - o_{i-1}(q))$$

The lower half of the Jacobian $J_\omega(q)$ is given as

$$J_\omega(q) = [J_{\omega_1}(q) \quad J_{\omega_2}(q) \quad J_{\omega_3}(q) \quad J_{\omega_4}(q) \quad J_{\omega_5}(q)]$$

in which the i^{th} column $J_{\omega_i}(q)$ is

$$J_{\omega_i}(q) = z_{i-1}(q)$$

As what is done in the forward kinematics, for any given configuration q, the homogeneous transformation matrix from the base frame to the i^{th} joint's is given as

$$T_{i-1}^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We can get $z_{i-1}(q)$ from the third column, $o_{i-1}(q)$ from the fourth column, and $o_e(q)$ from the fourth column of T_e^0.

Now, for a given configuration q, and joint velocity $\dot{q}$, we may write the forward velocity kinematics as

$$\xi = J(q)\dot{q}$$

in which ξ and $J(q)$ are given by

$$\xi = \begin{bmatrix} v_e^0 \\ \omega_e^0 \end{bmatrix} \qquad \text{and} \qquad J(q) = \begin{bmatrix} J_v(q) \\ J_\omega(q) \end{bmatrix}$$

There is a MATLAB code *FK_velocity.m* accomplishing the task listed in the Appendix 1 and attached to the file.

1.2 Inverse velocity kinematics

1.2.1 Inverse velocity kinematics

The inverse kinematics is about the find the joint velocities $\dot{q}$ that produce the desired end-effector velocity ξ in the given configuration q. The flowchart of completing inverse velocity kinematics is shown as Figure 1-2.

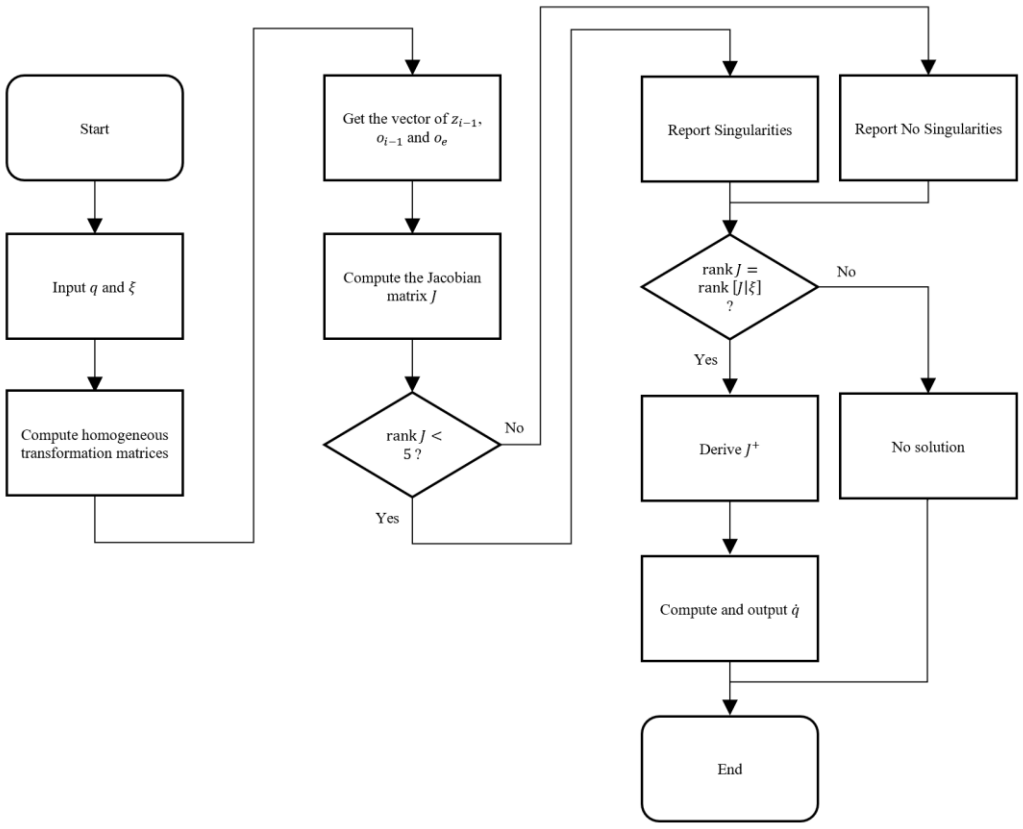

Figure 1-2 Flowchart of inverse velocity kinematics

The way to obtain the Jacobian matrix is the same as the above section. For our Lynx robot which do not have six valid joints, the Jacobian cannot be inverted. In this case there will be a solution to solving the inverse equation if and only if ξ lies in the range space of the Jacobian. This can be determined by the following simple rank test. A vector ξ belongs to the range of $J(q)$ if and only if

$$\operatorname{rank} J(q) = \operatorname{rank} [J(q) \mid \xi]$$

which is a standard result from linear algebra (SHV p151).

In our case, the Jacobian matrix(we will use J instead of $J(q)$ from here for concision) is a 6×5 matrix. We can solve for $\dot{q}$ using the left pseudoinverse of $J(q)$. To construct this pseudoinverse, we use the fact that when $J \in \mathbb{R}^{6\times5}$ and $\operatorname{rank} J = 5$, then $(J^T J)^{-1}$ exist. In this case $J^T J \in \mathbb{R}^{5\times5}$, and has $\operatorname{rank} 5$. Using this result, we can regroup terms to obtain

$$\begin{aligned} I &= (J^T J)^{-1} J^T J \\ &= ((J^T J)^{-1} J^T) J \\ &= J^+ J \end{aligned}$$

Here, $J^+ = (J^T J)^{-1} J^T$ is the left pseudoinverse of J.

Now we manage to solve $\dot{q}^+$ as a least-squares solution of the equation $J\dot{q} = \xi$.

In this particular case, we can write J in the form of singular value decomposition, as

$$J = U \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} V^T$$

in which $U \in \mathbb{R}^{6\times6}$ and $V \in \mathbb{R}^{5\times5}$ are orthogonal matrices, and $\Sigma \in \mathbb{R}^{5\times5}$ is a diagonal matrix of J in which σ is the singular values of J.

It is obvious that

$$J^+ = V \begin{bmatrix} \Sigma^{-1} \\ 0 \end{bmatrix} U^T$$

in which $\Sigma^{-1} \in \mathbb{R}^{5\times5}$ is a diagonal matrix and the element is σ^{-1}.

We want to find a $\dot{q}$ to minimized $\|J\dot{q} - \xi\|$. However, since U is an isometry, we have

$$\begin{aligned} \|J\dot{q} - \xi\| &= \|U\Sigma V^T \dot{q} - \xi\| \\ &= \|\Sigma V^T \dot{q} - U^T \xi\| \end{aligned}$$

Decomposing U as $[U_{1-5}, U_6]$, in which U_{1-5} is the first 5 columns of U and U_6 is the last column of U, regroup the terms as

$$\begin{aligned} \|\Sigma V^T \dot{q} - U^T \xi\| &= \|\Sigma V^T \dot{q} - [U_{1-5}, U_6]^T \xi\| \\ &= \left\| \begin{bmatrix} \Sigma V^T \dot{q} - {U_{1-5}}^T \xi \\ -{U_6}^T \xi \end{bmatrix} \right\| \\ &= \left\| \Sigma V^T \dot{q} - {U_{1-5}}^T \xi \right\| + \left\| -{U_6}^T \xi \right\| \end{aligned}$$

It is fairly obvious that $\left\|-{U_6}^T\xi\right\| \geq 0$. We need $\left\|\Sigma V^T\dot{q} - {U_{1-5}}^T\xi\right\| = 0$ to minimize the equation, thus

$$\begin{aligned}\dot{q}^+ &= (\Sigma V^T)^{-1}{U_{1-5}}^T\xi \\ &= V\Sigma^{-1}{U_{1-5}}^T\xi \\ &= V\begin{bmatrix}\Sigma^{-1} \\ 0\end{bmatrix}U^T\xi \\ &= J^+\xi\end{aligned}$$

which agrees with the slide in the lecture (16_velocityIK p34).

There is a MATLAB code *IK_velocity.m* accomplishing the task listed in the Appendix 1 and attached to the file.

1.2.2 Inverse linear velocity kinematics

In some cases, when the orientation of the end-effector does not matter, we just need the trajectory of the end-effector in the cartesian workspace. The task can be finished by setting the end-effector's linear velocity v_e^0 of the trajectory, then find the joint velocities $\dot{q}$ that produce the desired v_e^0 in the given configuration q. The flowchart of completing inverse linear velocity kinematics is shown as Figure 1-3.

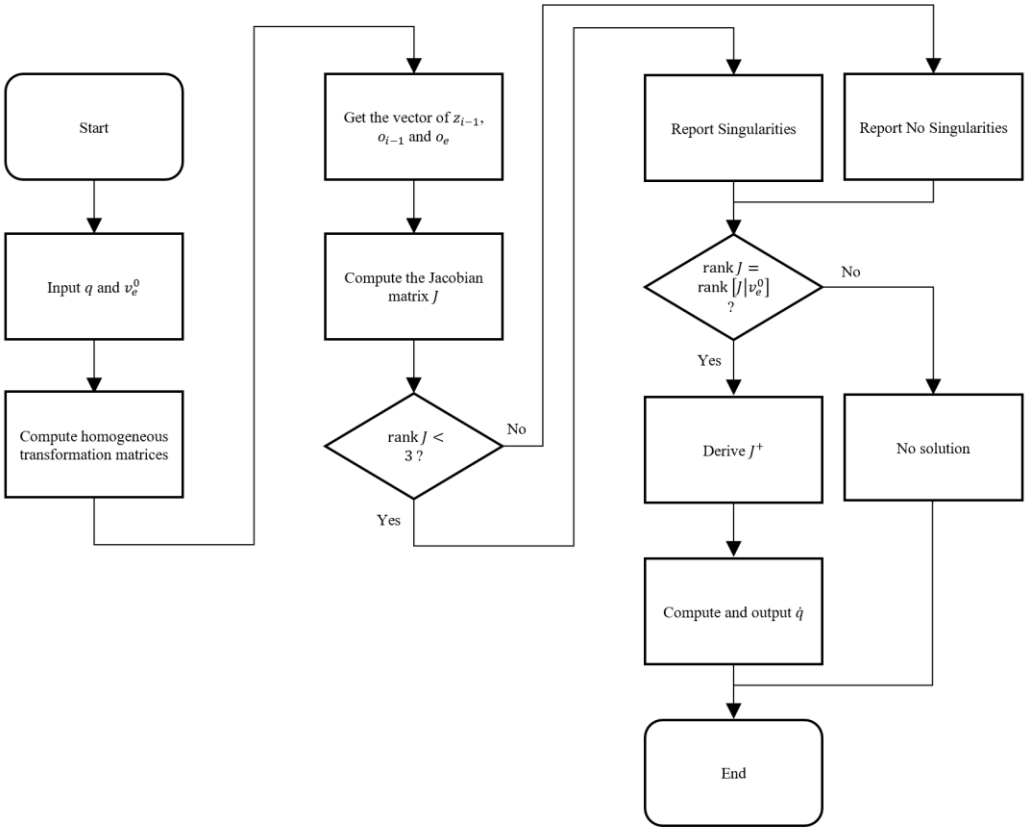

Figure 1-3 Flowchart of inverse linear velocity kinematics

The process is basically the same as inverse velocity kinematics stated above. There are some differences, however. In the previous inverse velocity kinematics, $J \in \mathbb{R}^{6\times5}$, we check the singularities by judge whether $rank\, J < 5$. In this inverse linear velocity kinematics, $J \in \mathbb{R}^{3\times5}$, we check the singularities by judge whether $rank\, J < 3$ instead.

In this case, the Jacobian matrix is a 3×5 matrix. We need solve for $\dot{q}$ using the right pseudoinverse rather than the left pseudoinverse of J. By the fact that $J \in \mathbb{R}^{3\times5}$ and when $rank\, J = 3$, then $(JJ^T)^{-1}$ exist. In this case $JJ^T \in \mathbb{R}^{3\times3}$, and has rank 3. Using this result, we can regroup terms to obtain

$$\begin{aligned} I &= JJ^T(JJ^T)^{-1} \\ &= J(J^T(JJ^T)^{-1}) \\ &= JJ^+ \end{aligned}$$

Here, $J^+ = (J^T(JJ^T)^{-1})$ is the right pseudoinverse of J.

It can be easily proved that $\dot{q}^+ = J^+\xi$ is still a solution to the equation $J\dot{q} = \xi$, shown as followed.

$$\begin{aligned} \xi &= J\dot{q} \\ &= JJ^+\xi \\ &= I\xi \\ &= \xi \end{aligned}$$

There is a MATLAB code *IK_linearvelocity.m* accomplishing the task listed in the Appendix 1 and attached to the file.

1.2.3 Velocity trajectory planning with orientation control

As what is stated above, $J\dot{q} = \xi$ has solutions iff $\text{rank}\, J = \text{rank}\, [J \mid \xi]$. In most case, however, for an arbitrary configuration q and an arbitrary ξ, $\text{rank}\, J$ would be 5 and $\text{rank}\, [J \mid \xi]$ would be 6, therefore there would be no solutions.

The physical interpretation is that our Lynx robot only has 5 DOF (joints), if you demand it to achieve a goal with 6 DOF, say, achieve the desired end-effector speed ξ, it would fail at the most of time. Therefore, we manage to make our desired ξ as an input of 5 DOF, in another words, make $\text{rank}\, \xi = 5$.

The symbolic representation of the Lynx robot is shown as Figure1-4. In which, we interpret the end-effector linear velocity v_e^0 as a linear combination of v_x, v_y, and v_z. We interpret the end-effector angular velocity ω_e^0 as a linear combination of ω_{z1}, ω_{z2}, and ω_{z5}, which are in the direction of z_1, z_2, and z_5, respectively. Thus, ξ is interpreted as a linear combination of v_x, v_y, v_z, ω_{z1}, ω_{z2}, and ω_{z5}, as

$$\begin{bmatrix} v_{x0} \\ v_{y0} \\ v_{z0} \\ \omega_{x0} \\ \omega_{y0} \\ \omega_{z0} \end{bmatrix} = \xi = A \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_{z1} \\ \omega_{z2} \\ \omega_{z5} \end{bmatrix}$$

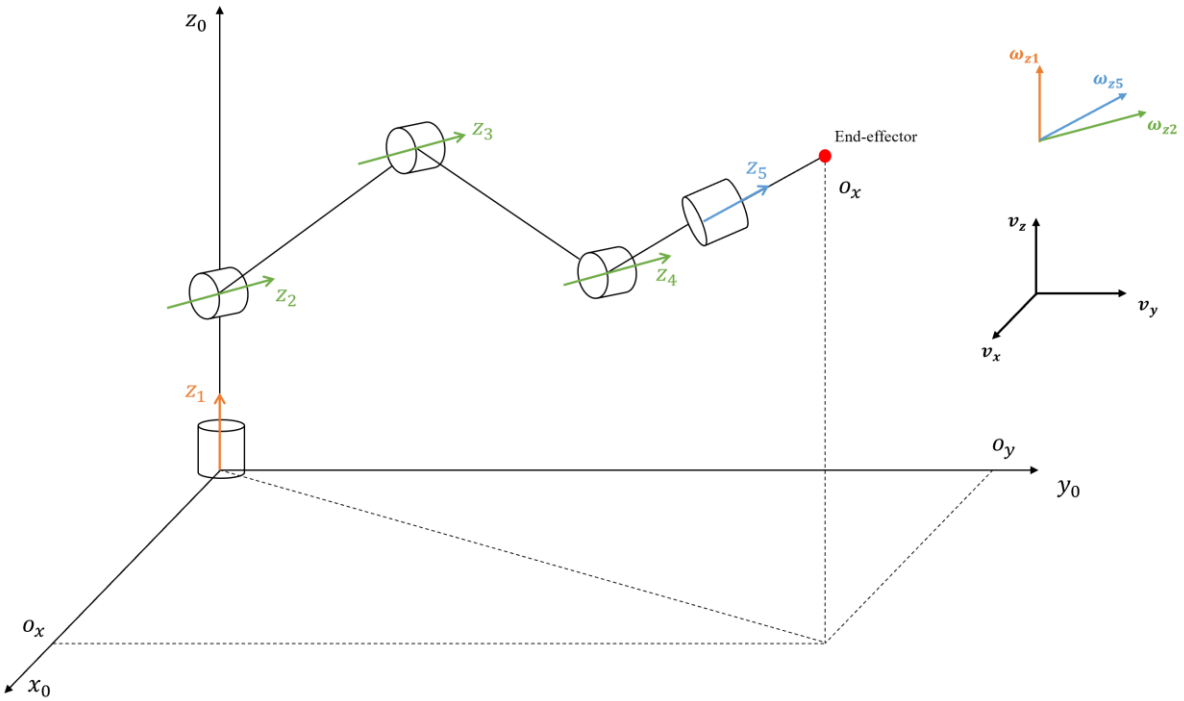

Figure 1-4 Symbolic representation of the robot

It is obvious that $rank\ [v_x\ v_y\ v_z\ \omega_{z1}\ \omega_{z2}\ \omega_{z5}]^T = 6$. In order to make rank $\xi = 5$, we need $rank\ A = 5$, in another words, one element in the $[v_x\ v_y\ v_z\ \omega_{z1}\ \omega_{z2}\ \omega_{z5}]^T$ cane be expressed as the linear combination of the other 5 elements. We call it the passive element.

We choose ω_{z1} as the passive element since it is obvious that ω_{z1} is related with v_x and v_y, in another word, we interpret ω_{z1} as the angular velocity created by v_x and v_y. The relationship can be derived as followed.

$$\begin{bmatrix} v_x \\ v_y \\ 0 \end{bmatrix} = \omega_{z1} \times o_e^0 = S(\omega_{z1})r = \begin{bmatrix} 0 & -\omega_{z1} & 0 \\ \omega_{z1} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix} = \begin{bmatrix} -\omega_{z1} o_y \\ \omega_{z1} o_x \\ 0 \end{bmatrix}$$

Then,

$$\omega_{z1} = \frac{v_y o_x - v_x o_y}{{o_x}^2 + {o_y}^2}$$

Thus, we set our input with 5 DOF as $\xi' = [v_x\ v_y\ v_z\ \frac{v_y o_x - v_x o_y}{{o_x}^2 + {o_y}^2}\ \omega_{z2}\ \omega_{z5}]^T$, and ξ can be projected from ξ' as

$$\begin{bmatrix} v_{x0} \\ v_{y0} \\ v_{z0} \\ \omega_{x0} \\ \omega_{y0} \\ \omega_{z0} \end{bmatrix} = \xi = B\xi' = \begin{bmatrix} B_v & 0 \\ 0 & B_\omega \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \frac{v_y o_x - v_x o_y}{{o_x}^2 + {o_y}^2} \\ \omega_{z2} \\ \omega_{z5} \end{bmatrix}$$

in which

$$B_v = I$$

$$B_\omega = \begin{bmatrix} 0 & -\dfrac{o_y}{\sqrt{{o_x}^2 + {o_y}^2}} & r_{13} \\ 0 & \dfrac{o_x}{\sqrt{{o_x}^2 + {o_y}^2}} & r_{23} \\ 1 & 0 & r_{33} \end{bmatrix}$$

in which

r_{13}, r_{23} and r_{33} are elements from $R_e^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$

Now, in most cases, such $\text{rank}\, J = \text{rank}\, [J \mid \xi]$ can be satisfied for an arbitrary configuration q, and an arbitrary ξ'. We can now solve the inverse velocity kinematics as $\dot{q}^+ = J^+ \xi$, and completing velocity trajectory planning with orientation control.

2 Evaluation

2.1 Start in the zero position and one of the joints moves

When the Lynx robot is in the zero position, the symbolic representation of it is shown as Figure 2-1.

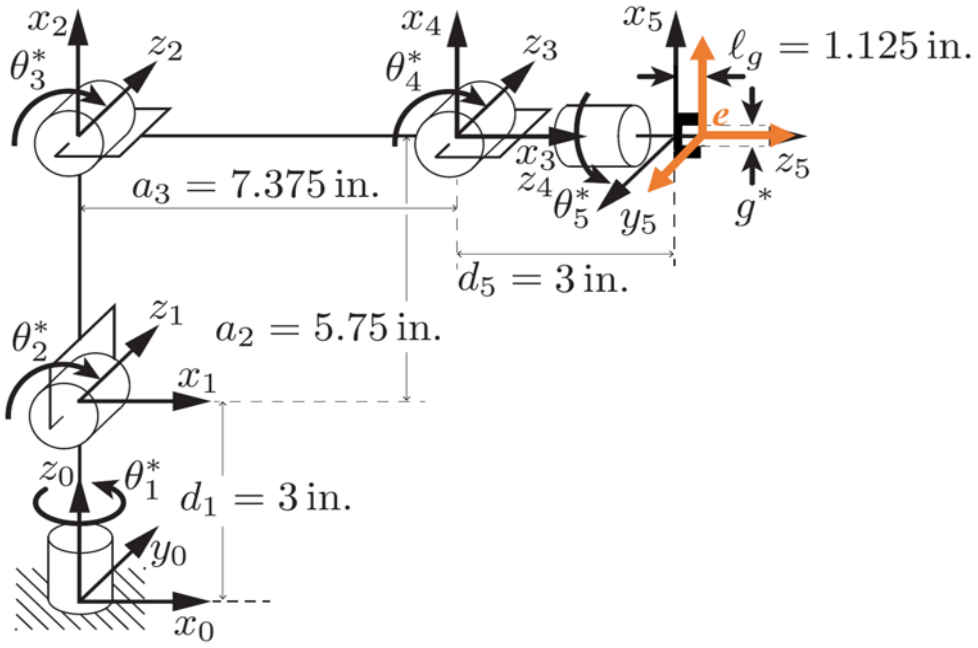

Figure 2-1 Symbolic representation of Lynx robot in zero position

When joint 1 moves at the angular speed of $\dot{q_1}$, it is obvious that the angular velocity $\omega_e^0 = [0 \quad 0 \quad \dot{q_1}]^T$.

The linear velocity can be obtained as

$$v_e^0 = \omega_e^0 \times (o_e - o_0) = \dot{q_1} S(k)(o_e - o_0) = [0 \quad (a_3 + d_5 + l_g)\dot{q_1} \quad 0]^T = [0 \quad 292.1\dot{q_1} \quad 0]^T.$$

Given the zero configuration $q = [0\ 0\ 0\ 0\ 0\ 0]$ and the joint velocity $\dot{q} = [\dot{q_1} 0\ 0\ 0\ 0\ 0]$, the command and the result of our FK is shown as followed, which meet the expectation.

```
>> q = [0 0 0 0 0 0];
>> syms qdot1 real;
>> qdot = sym([0 0 0 0 0 0]);
>> qdot(1) = sym('qdot1');
>> e_vel = FK_velocity(q,qdot)

 e_vel =

            0
(2921*qdot1)/10
            0
            0
            0
         qdot1
```

When joint 2 moves at the angular speed of $\dot{q}_2$, it is obvious that the angular velocity $\omega_e^0 = [0 \quad \dot{q}_2 \quad 0]^T$.

The linear velocity can be obtained as

$$v_e^0 = \omega_e^0 \times (o_e - o_1) = \dot{q}_2 S(j)(o_e - o_1) = [a_2\dot{q}_2 \quad 0 \quad -(a_3 + d_5 + l_g)\dot{q}_2]^T = [146.05\dot{q}_2 \quad 0 \quad -292.1\dot{q}_2]^T.$$

Given the zero configuration $q = [0\ 0\ 0\ 0\ 0\ 0]$ and the joint velocity $\dot{q} = [0\ \dot{q}_2\ 0\ 0\ 0\ 0]$, the command and the result of our FK is shown as followed, which meet the expectation.

```
>> q = [0 0 0 0 0 0];
>> syms qdot2 real;
>> qdot = sym([0 0 0 0 0 0]);
>> qdot(2) = sym('qdot2');
>> e_vel = FK_velocity(q,qdot)

e_vel =

 (2921*qdot2)/20
              0
-(2921*qdot2)/10
              0
           qdot2
              0
```

When joint 3 moves at the angular speed of $\dot{q}_3$, it is obvious that the angular velocity $\omega_e^0 = [0 \quad \dot{q}_3 \quad 0]^T$.

The linear velocity can be obtained as

$$v_e^0 = \omega_e^0 \times (o_e - o_2) = \dot{q}_3 S(j)(o_e - o_2) = [0 \quad 0 \quad -(a_3 + d_5 + l_g)\dot{q}_3]^T = [0 \quad 0 \quad -292.1\dot{q}_3]^T.$$

Given the zero configuration $q = [0\ 0\ 0\ 0\ 0\ 0]$ and the joint velocity $\dot{q} = [0\ 0\ \dot{q}_3\ 0\ 0\ 0]$, the command and the result of our FK is shown as followed, which meet the expectation.

```
>> q = [0 0 0 0 0 0];
>> syms qdot3 real;
>> qdot = sym([0 0 0 0 0 0]);
>> qdot(3) = sym('qdot3');
>> e_vel = FK_velocity(q,qdot)

e_vel =

              0
              0
-(2921*qdot3)/10
              0
           qdot3
              0
```

When joint 4 moves at the angular speed of $\dot{q}_4$, it is obvious that the angular velocity $\omega_e^0 = [0 \quad \dot{q}_4 \quad 0]^T$.

The linear velocity can be obtained as

$$v_e^0 = \omega_e^0 \times (o_e - o_3) = \dot{q}_4 S(j)(o_e - o_3) = [0 \quad 0 \quad -(d_5 + l_g)\dot{q}_4]^T = [0 \quad 0 \quad -104.775\dot{q}_4]^T.$$

Given the zero configuration $q = [0\ 0\ 0\ 0\ 0\ 0]$ and the joint velocity $\dot{q} = [0\ 0\ 0\ \dot{q}_4\ 0\ 0]$, the command and the result of our FK is shown as followed, which meet the expectation.

```
>> q = [0 0 0 0 0 0];
>> syms qdot4 real;
>> qdot = sym([0 0 0 0 0 0]);
>> qdot(4) = sym('qdot4');
>> e_vel = FK_velocity(q,qdot)

e_vel =

            0
            0
 -(4191*qdot4)/40
            0
         qdot4
            0
```

When joint 5 moves at the angular speed of $\dot{q}_5$, it is obvious that the angular velocity $\omega_e^0 = [\dot{q}_5 \quad 0 \quad 0]^T$.

It is also obvious that the linear velocity $v_e^0 = [0 \quad 0 \quad 0]^T$.

Given the zero configuration $q = [0\ 0\ 0\ 0\ 0\ 0]$ and the joint velocity $\dot{q} = [0\ 0\ 0\ 0\ \dot{q}_5\ 0]$, the command and the result of our FK is shown as followed, which meet the expectation.

```
>> q = [0 0 0 0 0 0];
>> syms qdot5 real;
>> qdot = sym([0 0 0 0 0 0]);
>> qdot(5) = sym('qdot5');
>> e_vel = FK_velocity(q,qdot)

e_vel =

    0
    0
    0
 qdot5
    0
    0
```

In sum, when the Lynx robot starts in zero position and only one of the joint moves, our FK meets the expectations of the corresponding velocity of the end effector.

2.2 Singularities issue

Project the arm in the plane formed by the links, some singularities configurations and are shown as Figure 2-2.

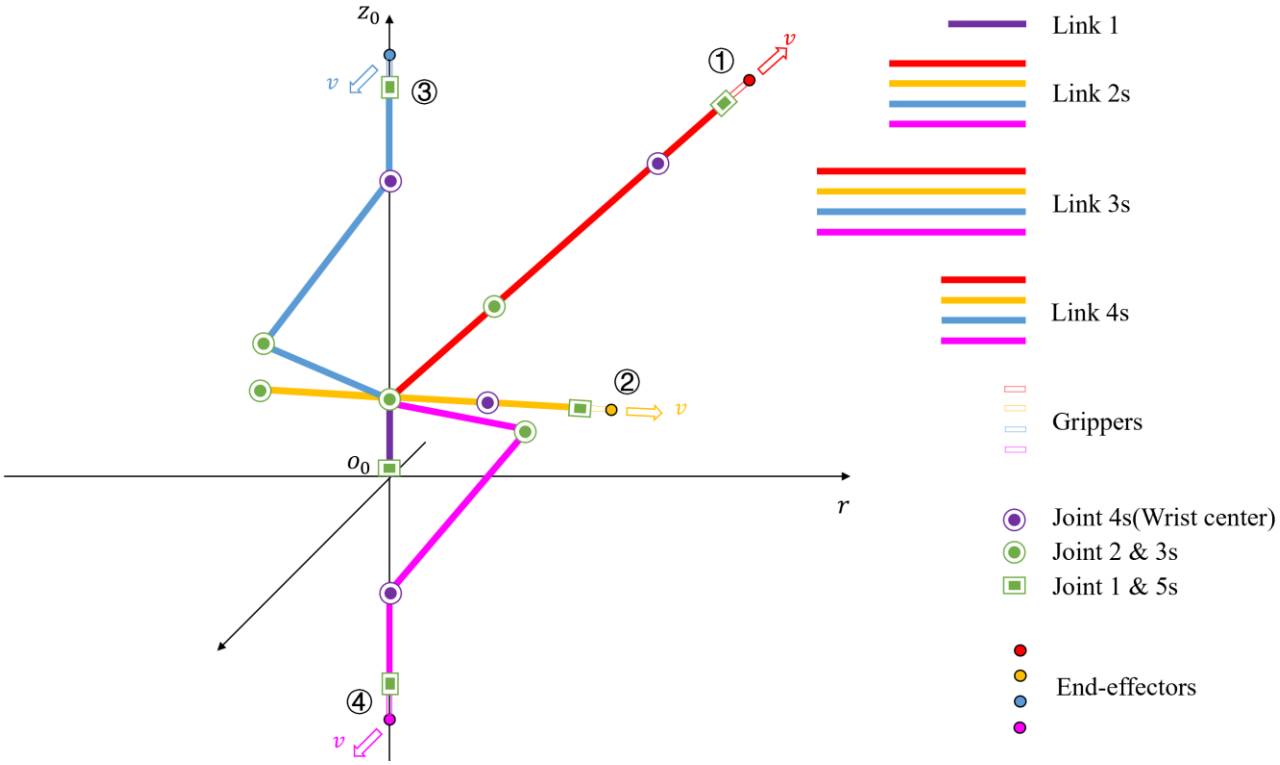

Figure 2-2 Singularities configurations on the plane formed by the links

There are infinite many singularity configurations, but it can be classified into 2 categories.

The first case is that the link 2, link3 link4 and the grippers are formed as a straight line in this plane, shown as ① and ② in Figure 2-2. In such configurations, the end-effector cannot move in the direction collinear with the links, which shown as v and v in Figure 2-2. The robot loses 1 DOF. Try to make such moves in this configurations would be no solutions to the inverse velocity kinematics.

Configuration in this case can be easily input into MATLAB to run the mathematical check whether $rank\ J < 5$ in such configurations, which result as $rank\ J = 4$. Then, call the inverse linear velocity kinematics function to make such moves, which result as no solutions thus meet the expectation.

The second case is that the end-effector and the wrist center both lie on the z_0 axis, shown as ③ and ④ in Figure 2-2. In such configurations, the end-effector cannot move in the direction perpendicular to the plane, which shown as v, and v in Figure 2-2. The robot loses 1 DOF. Try to make such moves in this configurations would be no solutions to the inverse velocity kinematics.

Use MATLAB to run inverse kinematics for wrist centers to get such configuration, then check whether $rank\ J < 5$ in such configurations, which result as $rank\ J = 4$. Then, call the inverse linear velocity kinematics function to make such moves, which result as no solutions thus meet the expectation.

The above results are shown as Figure 2-3 and Figure 2-4. There is a MATLAB code *singularities.m* accomplishing the task listed in Appendix 1 and attached to the file.

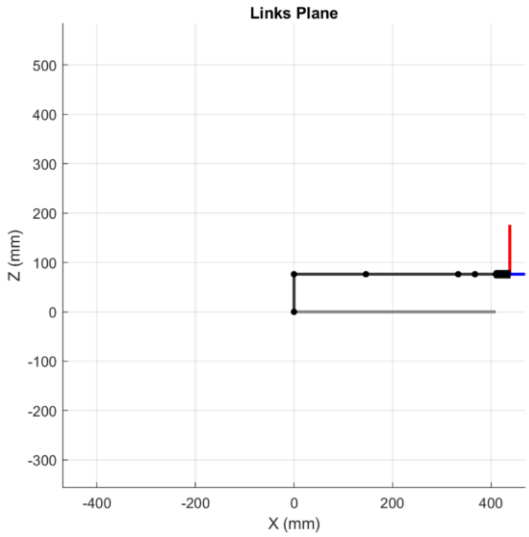

Figure 2-3 First singularity

```
>> rank(J)
ans =     4
>> qdot = IK_linearvelocity(q,[10;0;0;0;0;0]);
sigularities exist
Error using IK_linearvelocity (line 69)
e_vel is not a linear combination of the columns of J, no solutions exist
```

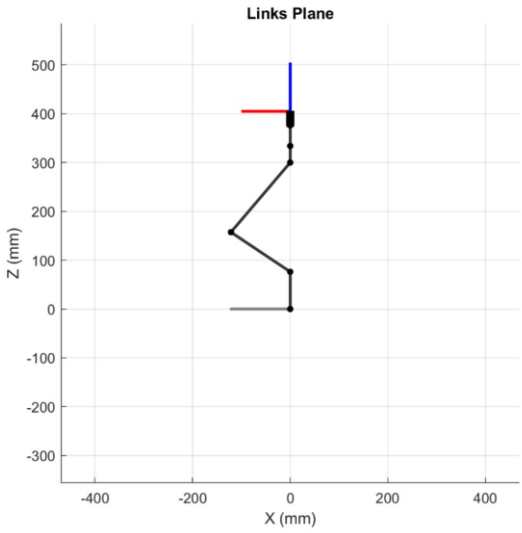

Figure 2-4 Second singularity

```
>> rank(J)
ans =     4
>> qdot = IK_linearvelocity(q,[0;10;0;0;0;0]);
sigularities exist
Error using IK_linearvelocity (line 69)
e_vel is not a linear combination of the columns of J, no solutions exist
```

2.3 End effector's trajectory when all joints move

Assume the robot starts in zero position, and all joints move at a constant and identical velocity. It is predictable that it would return to the zero position by the time all joints move a cycle.

Write a MATLAB code *lynxVelocitySim.m* showing the process of doing so from 4 different view. The yellow and pink bar is the visualized staff for linear velocity and angular velocity. The final result is shown as Figure 2-5, which the robot returns to zero position thus meet our expectation. There is a video (Video 1) showing the simulated process listed in Appendix 2.

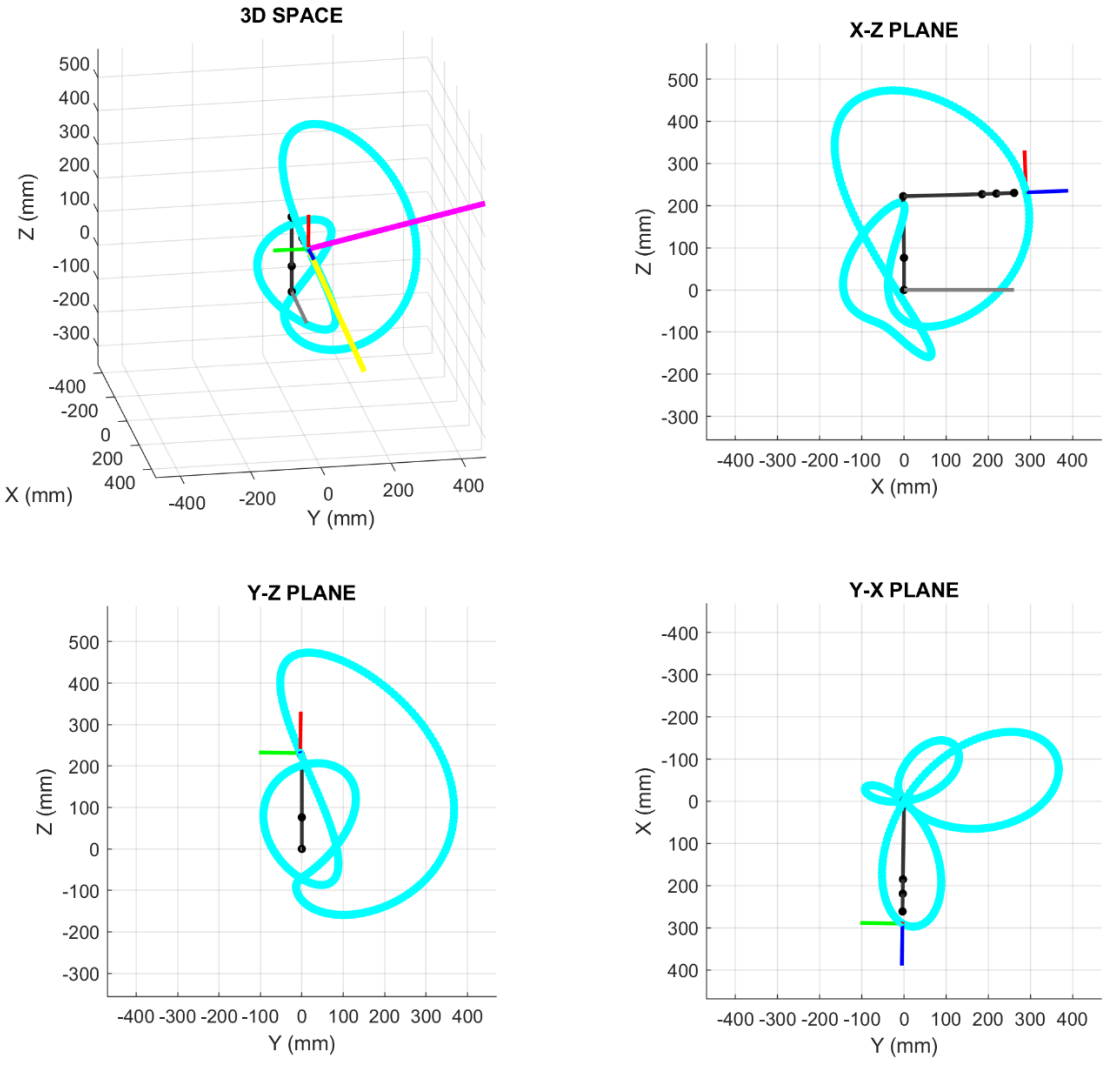

Figure 2-5 End-effector's trajectory when all joints move equally fast

2.4 Manage to move in a line

The parametric equations of a line in cartesian space are shown as

$$\begin{cases} x = at \\ y = bt \\ z = ct \end{cases}$$

Therefore, the parametric equations of the velocity tracing a line in cartesian space are the derivation of above, shown as

$$\begin{cases} \dot{x} = a \\ \dot{y} = b \\ \dot{z} = c \end{cases}$$

In this case, we don't care about end effector's orientation. Set $v_e^0 = [a\ b\ c]^T$ and a start configuration q to do the inverse linear velocity kinematics to iterate configurations and use *lynxVelocitySim.m* to plot.

The final result is shown as Figure 2-6, which is clearly a line thus meet our expectation. There is a video (Video 2) showing the simulated process and a video showing the physical robot doing so (Video 3) listed in Appendix 2.

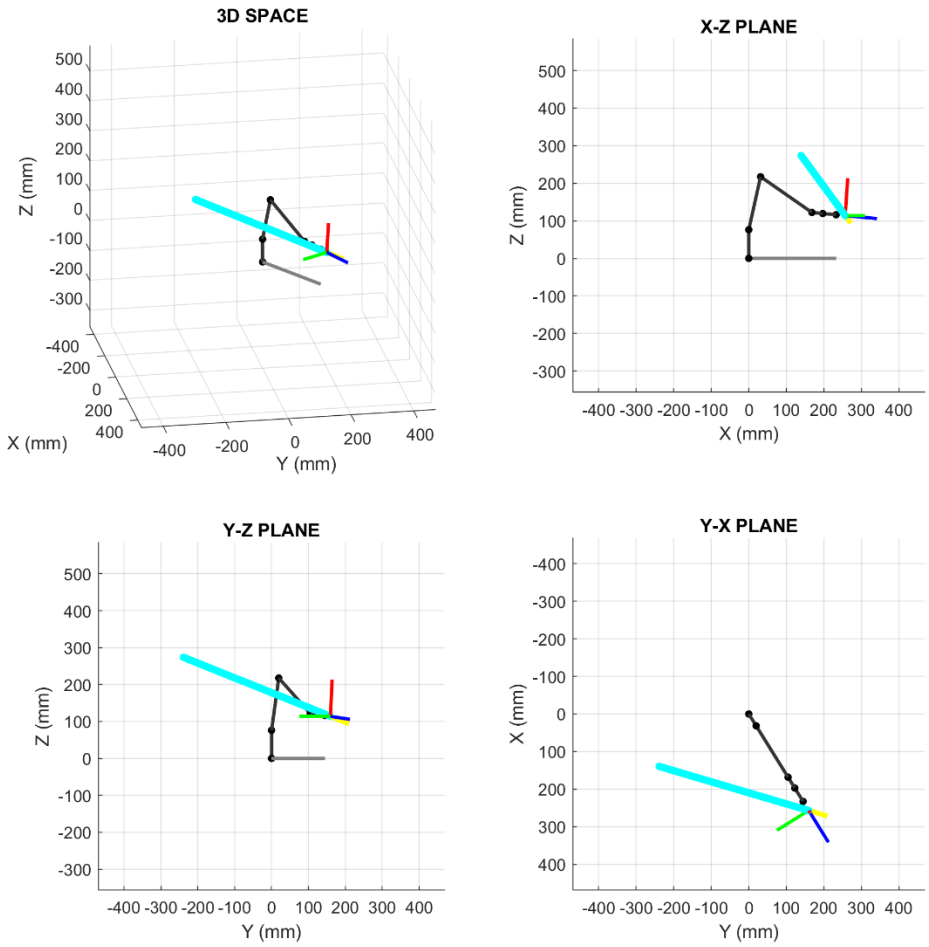

Figure 2-5 End-effector tracing in a line

2.5 Manage to move in a circle

Assume a circle in cartesian space with the center $[c_x \quad c_y \quad c_z]$, radius r, normal vector to the plane $n = [n_x \quad n_y \quad n_z]^T$. We set a vector lies in the plane $u = [u_x \quad u_y \quad u_z]^T = [-n_y \quad n_x \quad 0]^T$ orthogonal to n. Then set another vector liens in the plane $v = [v_x \quad v_y \quad v_z]^T = n \times u = \left[n_x n_z \quad n_y n_z \quad -n_x^2 - n_y^2\right]^T$. Normalize these two vectors as $\tilde{u} = [\widetilde{u_x} \quad \widetilde{u_y} \quad \widetilde{u_z}]^T$ and $\tilde{v} = [\widetilde{v_x} \quad \widetilde{v_y} \quad \widetilde{v_z}]^T$. We can get the parametric equations of this circle as

$$\begin{cases} x = c_x + r(\widetilde{u_x} cost + \widetilde{v_x} sint) \\ y = c_y + r(\widetilde{u_y} cost + \widetilde{v_y} sint) \\ z = c_z + r(\widetilde{u_z} cost + \tilde{v}_z sint) \end{cases}$$

Therefore, the parametric equation of the velocity in cartesian space is the derivation of above, shown as

$$\begin{cases} \dot{x} = r(-\widetilde{u_x} sint + \widetilde{v_x} cost) \\ \dot{y} = r(-\widetilde{u_y} sint + \widetilde{v_y} cost) \\ \dot{z} = r(-\widetilde{u_z} sint + \tilde{v}_z cost) \end{cases}$$

We also don't care about orientation in this case. Set the parameters and a start configuration q to do the inverse linear velocity kinematics to iterate configurations and use *lynxVelocitySim.m* to plot. The final result is shown as Figure 2-7, which is clearly a circle thus meet our expectation. There is a video (Video 4) showing the simulated process and a video showing the physical robot doing so (Video 5) listed in Appendix 2.

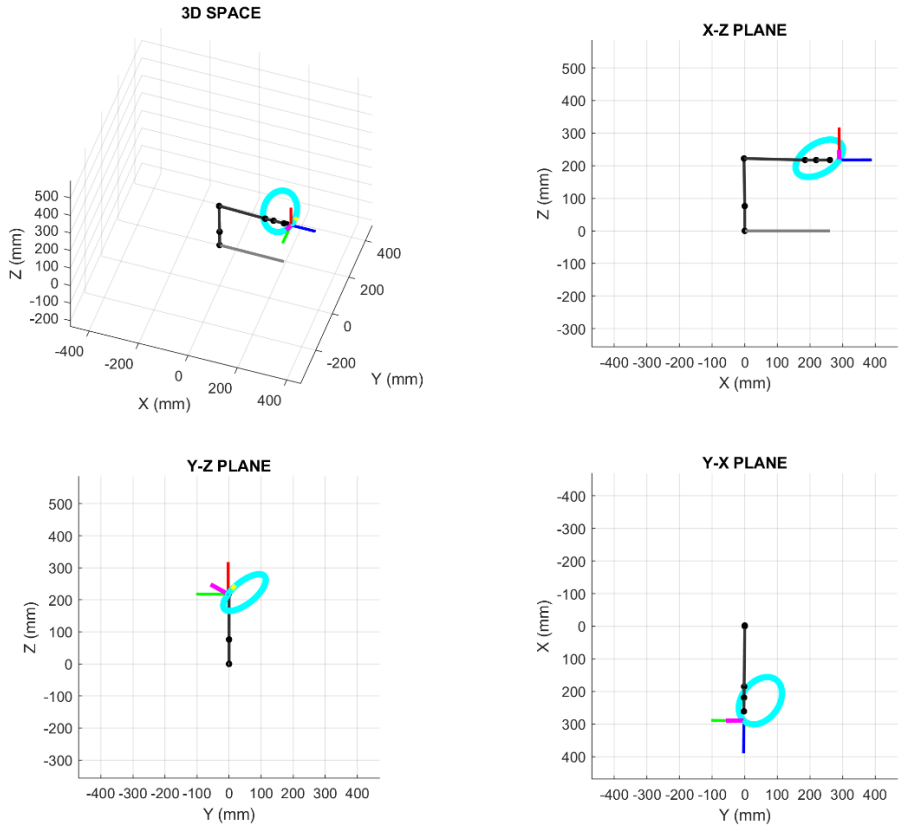

Figure 2-7 End-effector tracing in a circle

2.6 Control orientation when tracing in a line

2.6.1 Make the orientation fixed in the plane formed by the links

In this case, we want to control the orientation of the end-effector when it is tracing a line. Given a start configuration q, the start pose of the robot in 3D space and in the plane formed by links is shown as Figure 2-8.

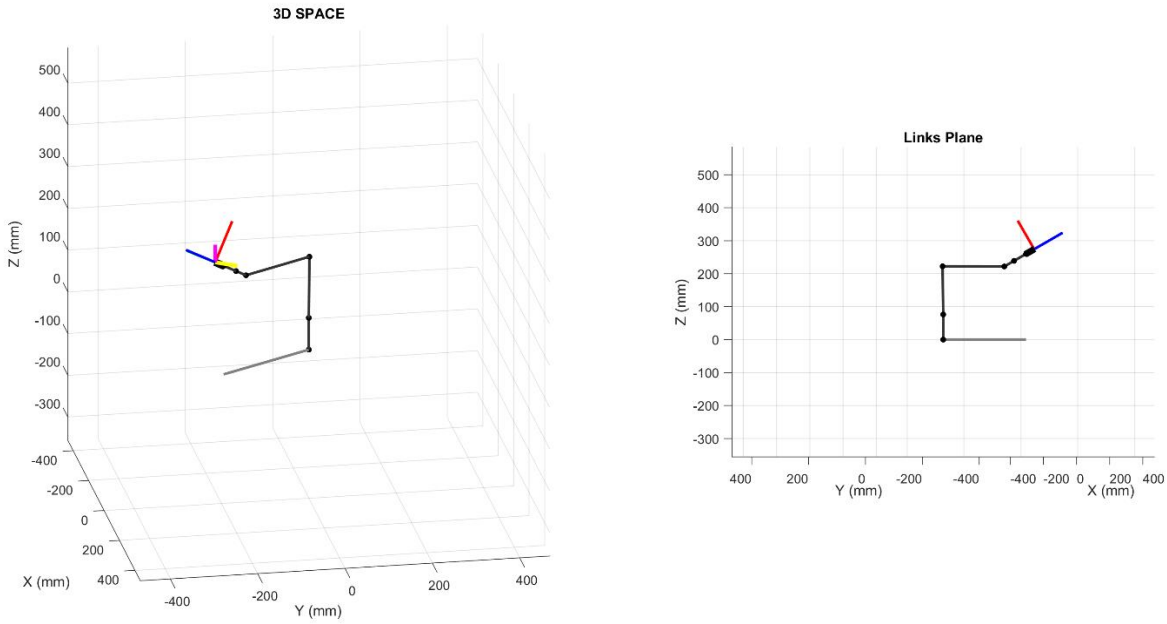

Figure 2-8 Robot in the start position

As what is stated in section 1.2.3, we set our desired velocity $\xi' = [v_x\ v_y\ v_z\ \frac{v_y o_x - v_x o_y}{{o_x}^2 + {o_y}^2}\ \omega_{z2}\ \omega_{z5}]^T$.

It is to say, in most case, what we can control about the orientation is the angular velocity in the direction of z_2, and the angular velocity in the direction of z_5. When we set $\omega_{z2} = 0$ and $\omega_{z5} = 0$, in the view of the plane formed by links, the orientation shall be fixed.

Write MATLAB codes *plotLynxO.m* show the process of doing so. Then write *plotLynxLinksPlane.m* to fix the view from the plane formed by links to make the observation clearer. The final result is shown as Figure 2-9, in which the orientation is thoroughly the same from the start position in the plane formed by links, thus meet our expectation. There is a video (Video 6) showing the simulated process and a video showing the physical robot doing so (Video 7) listed in Appendix 2.

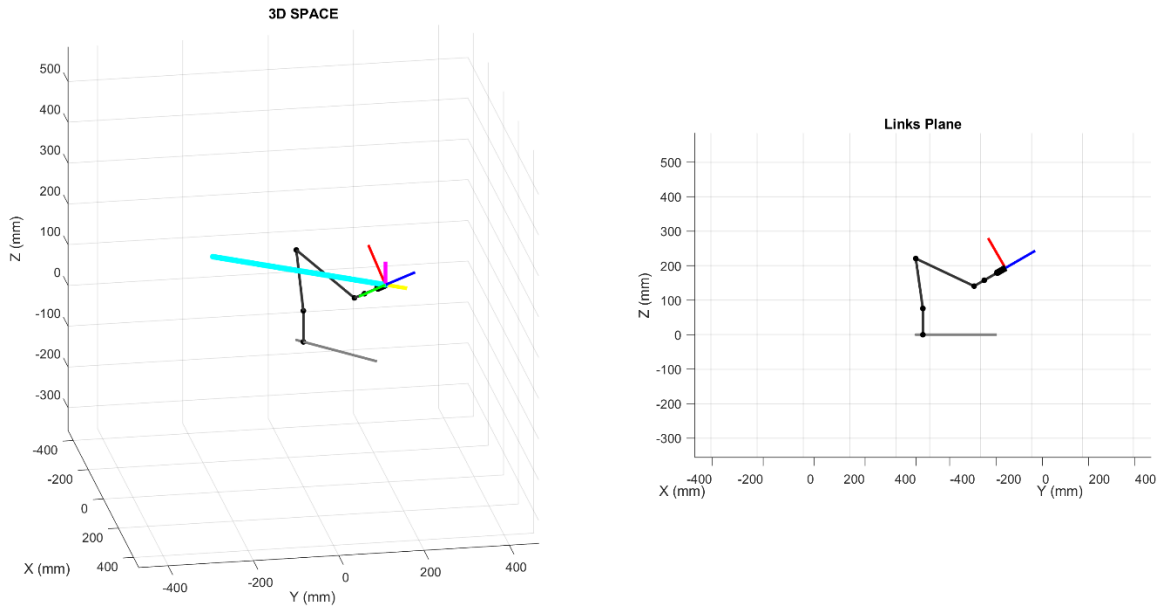

Figure 2-9 Robot in the end position

2.6.2 Make z_e rotate in the plane formed by the links

When we set ω_{z2} a constant speed and $\omega_{z5} = 0$, in the view of the plane formed by links, z_e shall be rotating in this process.

Run *plotLynxO.m* and *plotLynxLinksPlane.m* to observe this process. The final result is shown as Figure 2-10, in which z_e is thoroughly rotating the whole time in the plane formed by links, thus meet our expectation. There is a video (Video 8) showing the simulated process listed in Appendix 2.

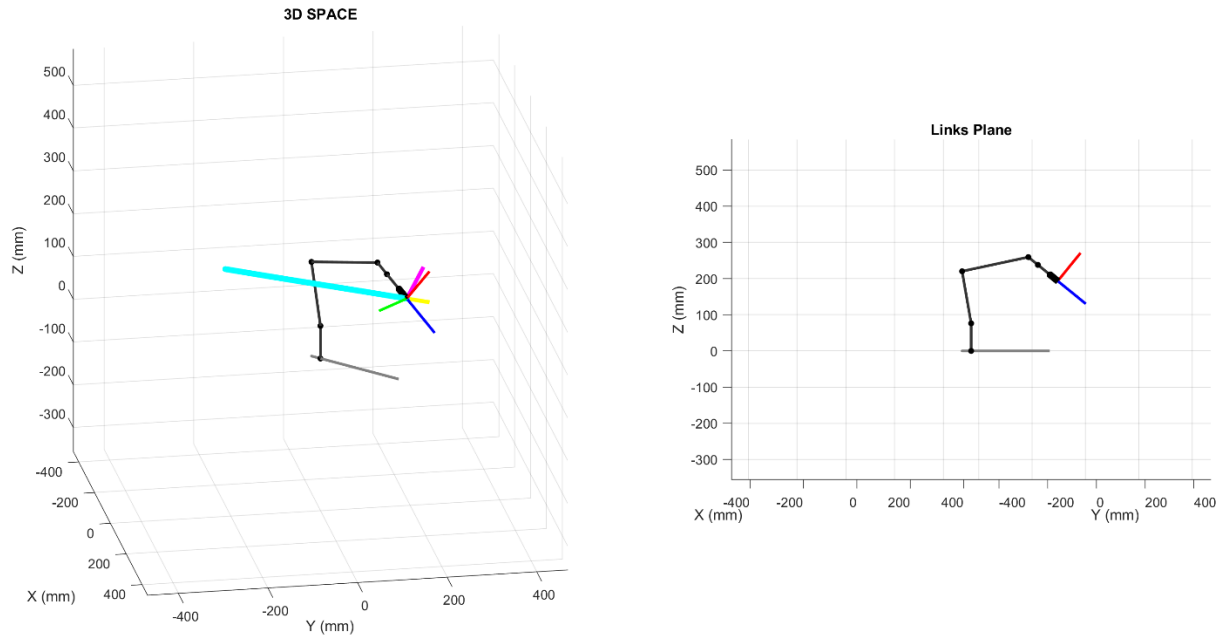

Figure 2-10 Robot in the end position

2.6.3 Make the end-effector rotate around z_e in the plane formed by the links

When we set $\omega_{z2} = 0$ and ω_{z5} a constant speed, in the view of the plane formed by links, the end-effector shall be rotating around z_e in this process.

Run *plotLynxO.m* and *plotLynxLinksPlane.m* to observe this process. The final result is shown as Figure 2-11, in which the end-effector is rotating around z_e the whole time in the plane formed by links, thus meet our expectation. There is a video (Video 9) showing the simulated process listed in Appendix 2.

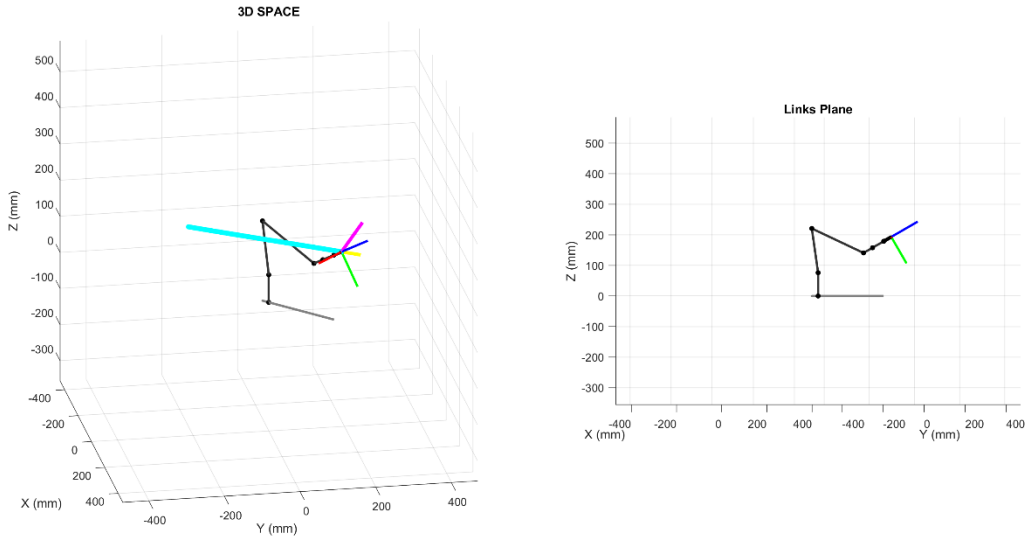

Figure 2-11 Robot in the end position

2.7 Make the orientation literately fixed in some trajectory

There are some particular cases that the orientation of the end-effector can be fixed thorough the whole trajectory, and it can be classified into 2 categories.

The first case is that when the joint 1 of the robot is fixed, then the robot becomes a planar arm. It is obvious that the orientation can be fixed if the robot is a planar arm, therefore we can just set our input as

$$\xi = \begin{bmatrix} v_{x0} \\ v_{y0} \\ v_{z0} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We manage to make the robot draw a love heart, which is the combination of two $\frac{3}{8}$ circles and two lines, with the orientation fixed. The final result is shown as Figure 2-12, in which the end-effector's orientation is thoroughly fixed in the trajectory thus meet out expectation. We noted the trajectory red because it is a love heart. There is a video (Video 10) showing the simulated process and a video (Video 11) showing the physical robot's moving listed in Appendix 2.

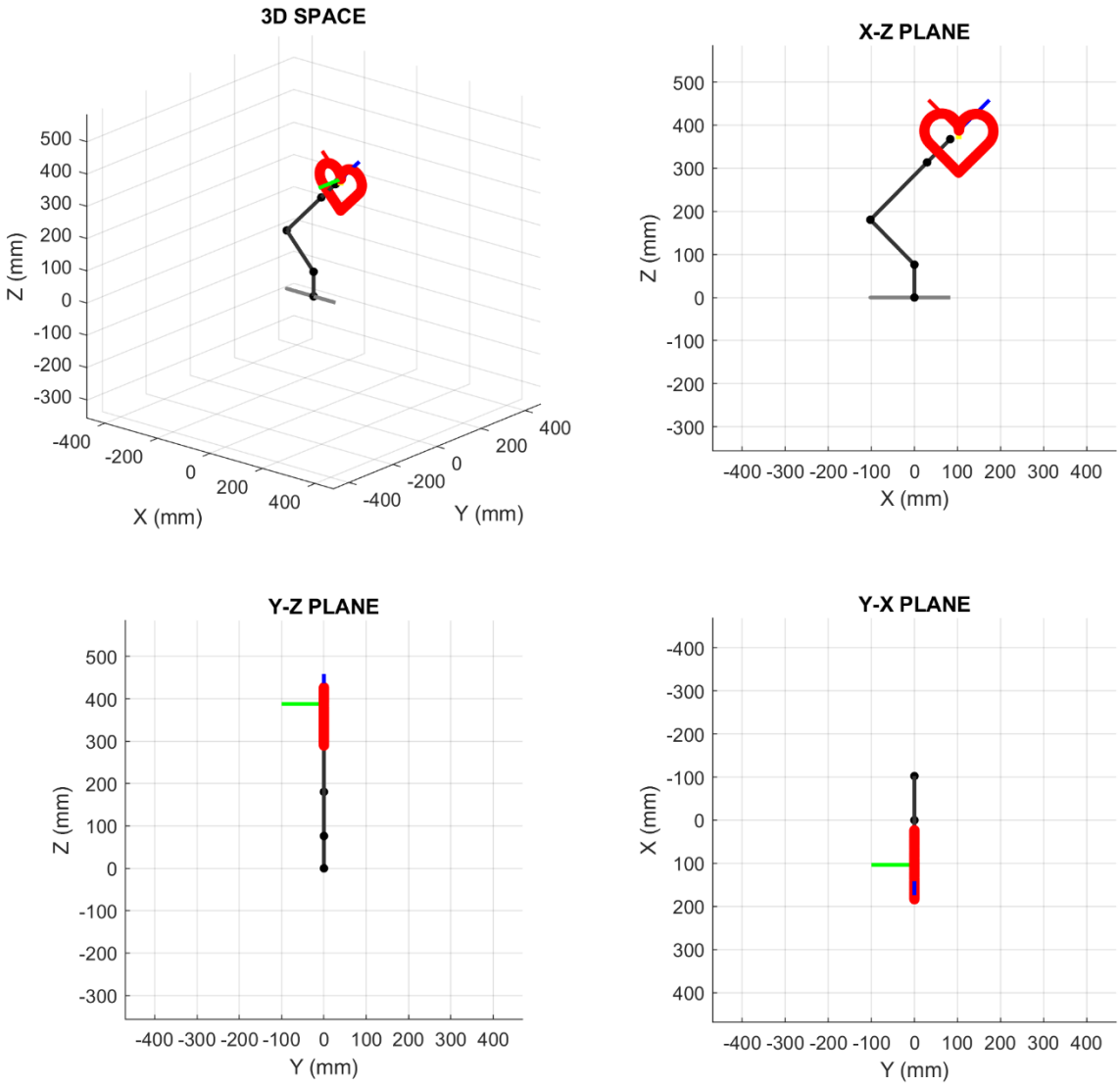

Figure 2-12 Drawing a love heart when joint 1 is fixed

The second case is that when the joint 1 is moving, if the end-effector is point collinear with the z_0 axis ($z_e \parallel z_0$), the orientation of the end-effector can be fixed because the angular velocity caused by joint 1 can be compensated by the angular velocity cause by joint 5. We can also set our input as

$$\xi = \begin{bmatrix} v_{x0} \\ v_{y0} \\ v_{z0} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Likewise, we manage to make the robot draw a love heart, with the orientation fixed. The final result is shown as Figure 2-13, in which the end-effector's orientation is thoroughly fixed in the trajectory. There is a video (Video 12) showing the simulated process a video (Video 13) showing the physical robot's moving listed in Appendix 2.

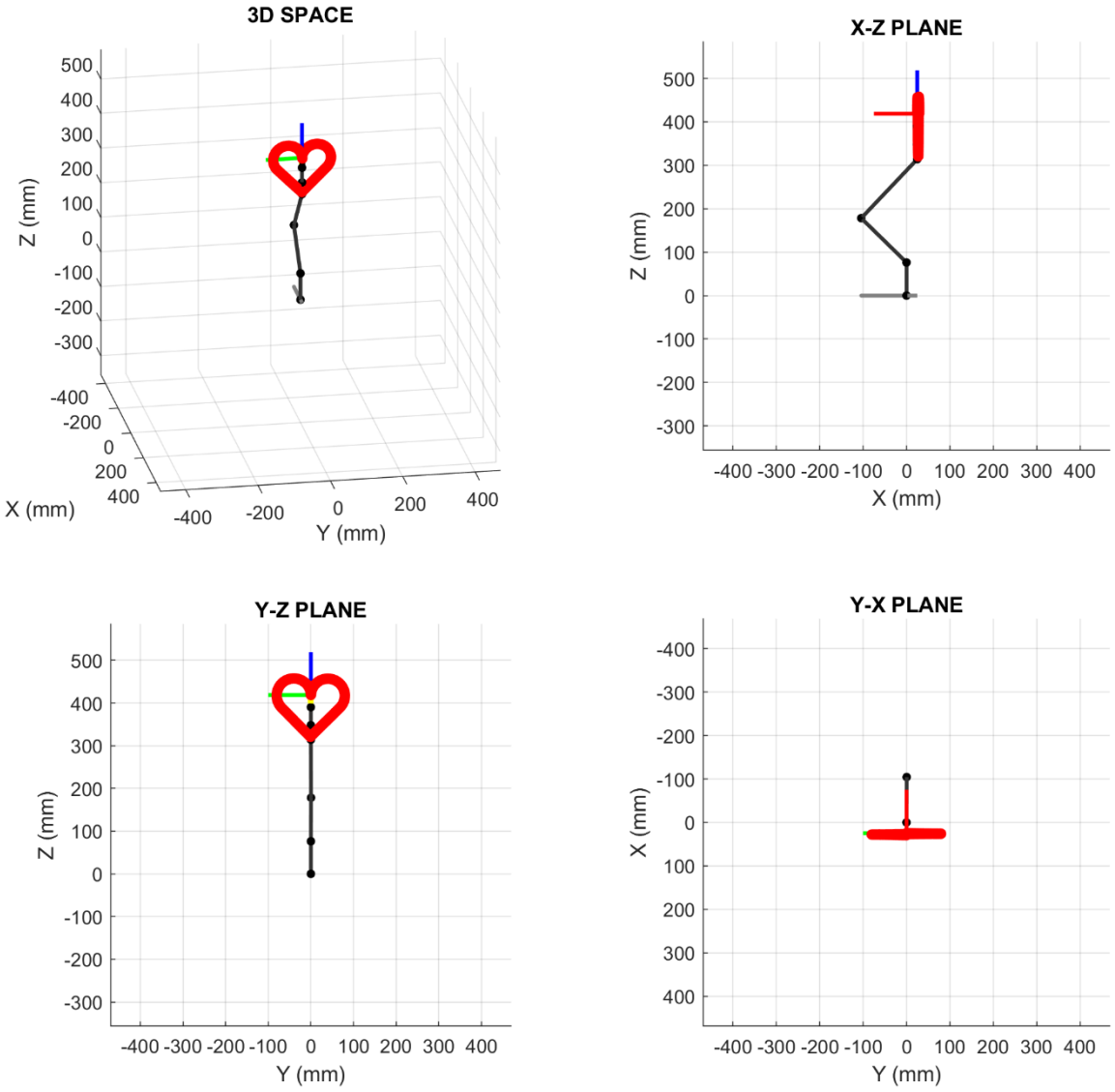

Figure 2-13 Drawing a love heart when joint 2 is fixed

There is a MATLAB code *runme.m* recording the command for the above trajectories listed in Appendix 1 and attached to the file.

3 Analysis

3.1 Compare the desired result and the outputted result in the evaluation cases

We test the IK like what we do to test FK in section 2.1. The command and result is shown as followed, which all meet our expectations.

```
>> q=[0 0 0 0 0 0];
>> qdot=[1 0 0 0 0 0];
>> e_vel = FK_velocity(q,qdot);
>> vpa(IK_velocity(q,e_vel),3)
No sigularity, there is only solution for e_vel
ans =

[ 1.0, 0, 0, 0, 0, 0]

>> qdot=[0 1 0 0 0 0];
>> e_vel = FK_velocity(q,qdot);
>> vpa(IK_velocity(q,e_vel),3)
No sigularity, there is only solution for e_vel
ans =

[ 0, 1.0, -9.66e-5, 9.22e-5, 0, 0]

>> qdot=[0 0 1 0 0 0];
>> e_vel = FK_velocity(q,qdot);
>> vpa(IK_velocity(q,e_vel),3)
No sigularity, there is only solution for e_vel
ans =

[ 0, 0, 1.0, 9.22e-5, 0, 0]

>> qdot=[0 0 0 1 0 0];
>> e_vel = FK_velocity(q,qdot);
>> vpa(IK_velocity(q,e_vel),3)
No sigularity, there is only solution for e_vel
ans =

[ 0, 0, -3.3e-5, 1.0, 0, 0]

>> qdot=[0 0 0 0 1 0];
>> e_vel = FK_velocity(q,qdot);
>> vpa(IK_velocity(q,e_vel),3)
No sigularity, there is only solution for e_vel
ans =

[ 0, 0, 0, 0, 1.0, 0]
```

Besides, as what is discussed in the Evaluation part, all our outputted result meets our expectation, it can be concluded that our forward and inverse velocity kinematics is reasonable enough for this robot.

3.2 The good-at movements and bad-at movements

In the simulated robots, we consider the manipulability as the ability of the movements of the Lynx robot. It is clear that when the configuration is away from singularities, the movement ability of the robot is stronger then the configuration near singularities, since it has a some ability to move in some direction when the configuration near singularities.

Assume all joint are moving at the constant speed. Draw an ellipsoid in the end-effector to show the ability of the movement, in which the length of each axis reflects the generates velocities in the axis' direction, respectively. The result are shown as Figure 3-1 and Figure 3-2, in which the volume of the ellipsoid when the configuration is away from singularities is greater than the one near singularities, thus meet our expectation.

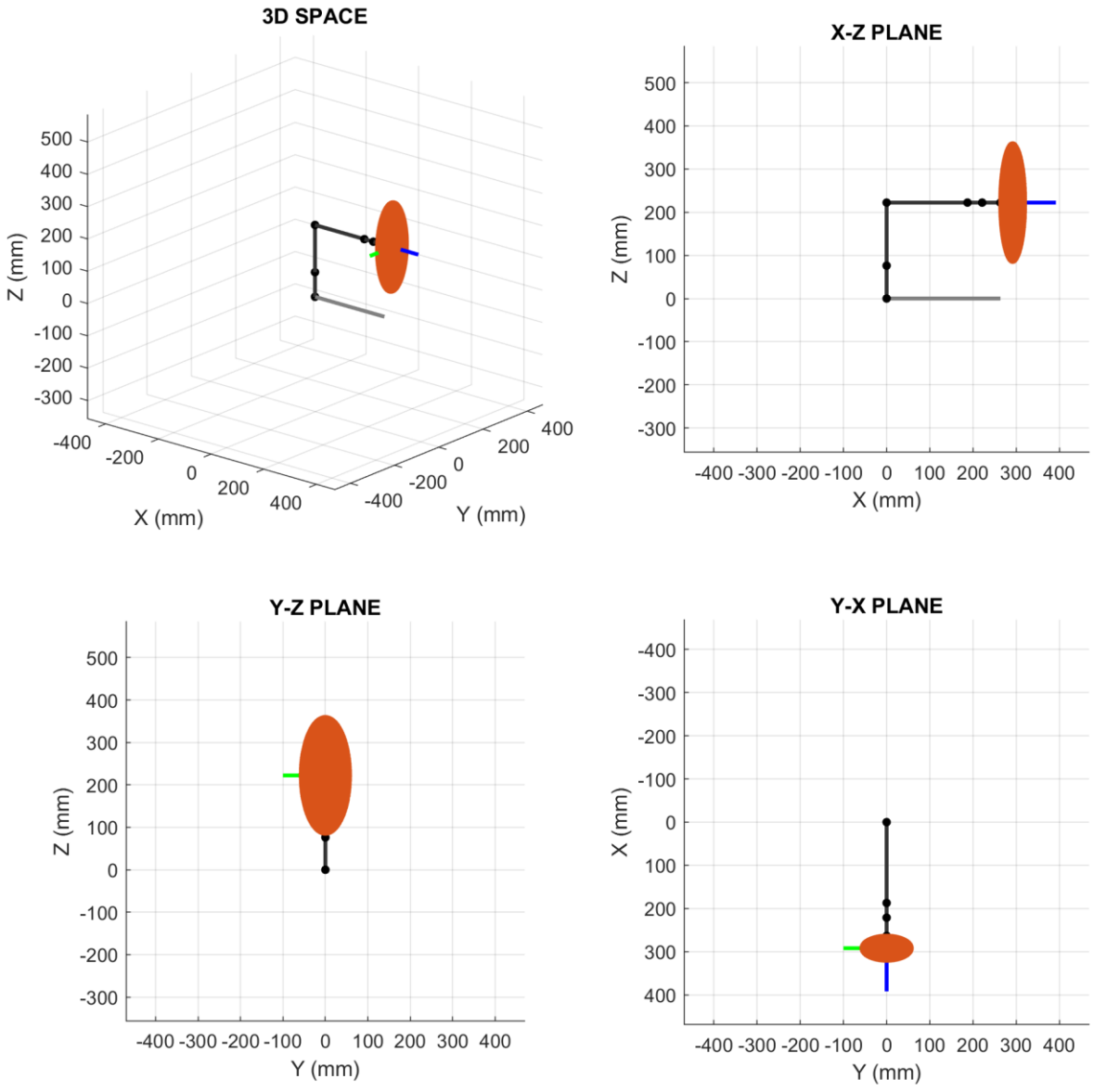

Figure 3-1 The manipulability of the robot in zero configuration

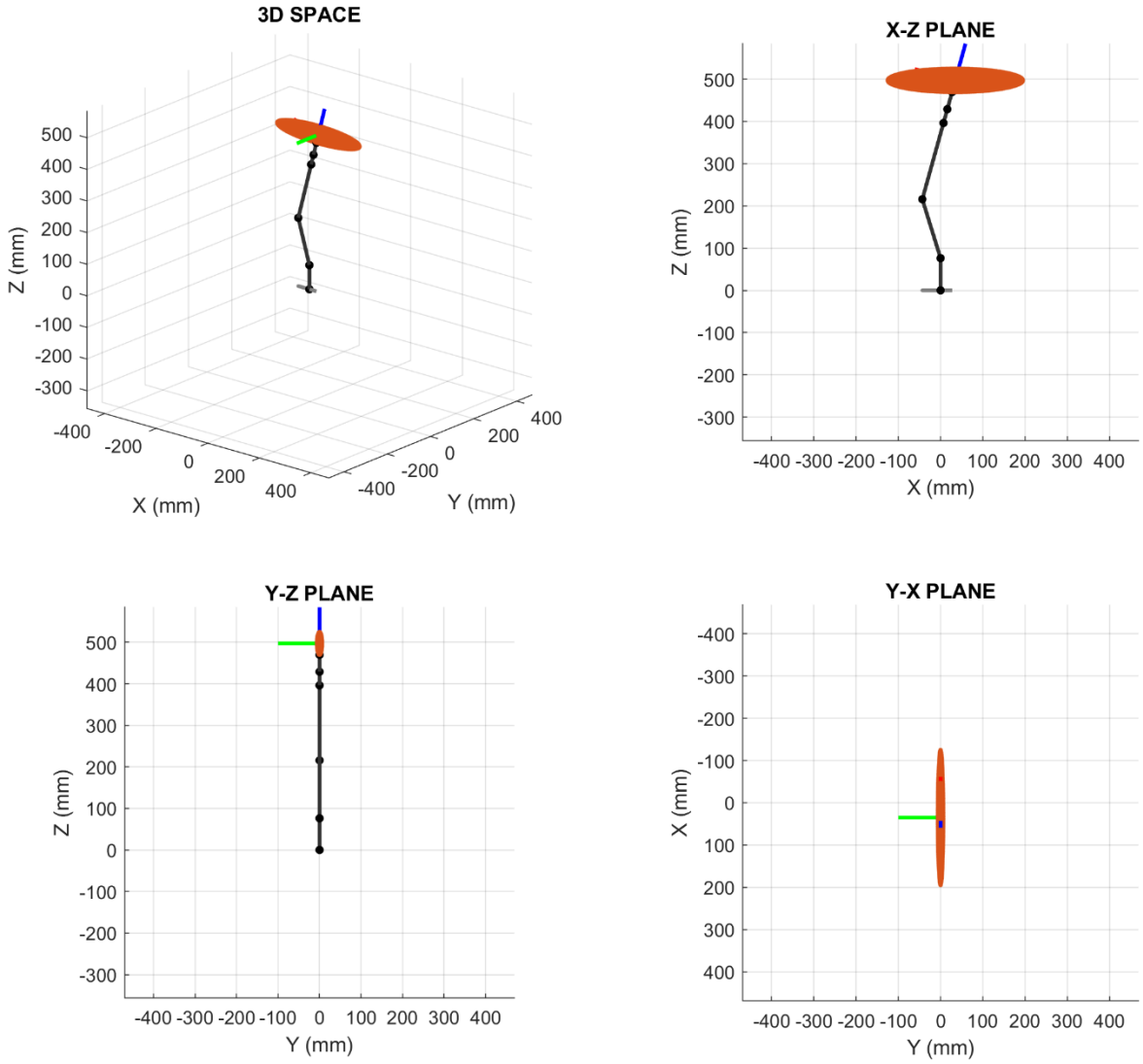

Figure 3-2 The manipulability of the robot near singular configuration

For the physical robots, expect the singularity, we have to also consider about some other issues like the physical shape of the robot, the errors caused by gravity and frictions. Also, the speed control function of the physical robot is not very good. The movement is bad for a relatively large range of the trajectory, the torque caused by gravity increases and iterates along the path, then the trajectory may not perform at what we expect. However, for some certain easy tasks like goes a straight line in a horizonal level, it may perform well.

3.3 Compare the velocity control with position control

For some tasks that velocity is not very essential and what all we need is the trajectory and don't care how the robot moves between points, then position control is easier.

However, if some specific tasks need the robot to go under a certain speed, like for some very precise operations or some very smooth movement is required, then velocity control is preferred.

Practically, when we manipulate arm robot to finish some certain tasks, i.e. grasp an object, when the arm robot moves from initial configuration to get close to the object, we usually use position control, while the arm robot is close enough to the object and about to get touched, then the velocity control is needed.

Besides, the velocity control can help us avoid inverse kinematics, which is way more complex than inverse velocity kinematics. If initial configuration is known, and the task is asked to plan the end-effector in the trajectory where the velocity parametric function can be easily obtained, i.e. tracing a line or circle, the velocity control is easier then position control.

Appendix 1 Code Overview

1. *calculateFK_sol.m* predefined compute forward kinematics
2. *FK_velocity.m* forward velocity kinematics function
3. *IK_linearvelocity.m* inverse linear velocity kinematics function
4. *IK_velocity.m* inverse velocity kinematics function
5. *lynxInitializeHardware.m* predefined lynx function
6. *lynxServo.m* predefined lynx function
7. *lynxServoSim.m* predefined lynx function
8. *lynxStart.m* modified lynx function
9. *lynxVelocityPhysical.m* predefined lynx function
10. *lynxServoPhysical.m* predefined lynx function
11. *lynxVelocitySim.m* simulate velocity control
12. *lynxVelocitySimO.m* simulate velocity control with orientation control
13. *manipulability.m* show the manipulability in different configurations
14. *plotLynx.m* plot simulated configuration in 3D space (modified)
15. *plotLynx1.m* plot simulated configuration in X-Z plane
16. *plotLynx2.m* plot simulated configuration in Y-Z plane
17. *plotLynx3.m* plot simulated configuration in Y-X plane
18. *plotLynxLinksPlane.m* plot simulated configuration in the plane formed by links
19. *plotLynxO.m* plot simulated configuration in 3D space
20. *runme.m* recording the comment for trajectory planning
21. *runVelocityPhysical.m* recording the comment for trajectory in physical robot
22. *singularity.m* show singularity issue

Appendix 2 Video Overview

1 https://youtu.be/6S4D6-ESltM	all joints move at constant speed
2 https://youtu.be/sY-L1I0ORz4	tracing a line in simulated robot
3 https://youtu.be/DFu8r1MVbDY	tracing a line in physical robot
4 https://youtu.be/PLZ3o4Qju9g	tracing a circle in simulated robot
5 https://youtu.be/2Xd3F_iwPqA	tracing a circle in physical robot
6 https://youtu.be/hQxt3sqVOxU	orientation is fixed in the links plane simulation
7 https://youtu.be/mpbzwq_Vn2I	orientation is fixed in the links plane physical
8 https://youtu.be/ozhh9cXtKWw	make ze rotate along the trajectory
9 https://youtu.be/IxSt5_u0FV0	rotate around ze along the trajectory
10 https://youtu.be/3s3LJjDZ_Lk	simulated love heart when joint 1 is fixed
11 https://youtu.be/1s8UPiSeBpY	physical love heart when joint 1 is fixed
12 https://youtu.be/Rz1PAjbeQkA	simulated love heart when joint 1 is moving
13 https://youtu.be/8XFUMj64LOM	physical love heart when joint 1 is moving