# Robotics Mini-Proj 3

Yuxin Hu

October 2021

## 1 Cover Page

**Name:** Yuxin Hu
**Course Number:** CSCI-4480
**Date:** Oct.17.2021

*I, Yuxin Hu, certify that the following work is my own and completed in accordance with the academic integrity policy as described in the Robotics I course syllabus.*

**Link to Github:** https://github.com/MoritomoNozomi/Robotics-I-Class-Project

# 2 Summary

In this mini-project, the algorithm for stereographic projection was implemented, and used to convert a planar S letter shape into a 3D shape on a sphere surface. Three representations of angles in 3D space (Unit Quaternion, Yaw-Pitch-Roll, and Angle-Axis Product) were discussed, and the algorithm to convert from R were implemented. The forward kinematics (POE, SDH, MDH) and inverse kinematics (Subproblem Decomposition Method) in 3D space was implemented. And finally, everything combined made it possible to control an ABB IRB1200/0.9 robot arm to follow a curve in 3D space.

# 3 Technical Content

## Part 1

**Description of the problem:**

(a) Generate the stereographic projection of the S letter curve provided.

(b) Represent the end effector orientation along the curve using:

1.Unit quaternion

2.yaw-pitch-roll ZYX

3.Angle-axis product.

**Derivation of the solution:**

(a) As discussed in the lecture, to find the stereographic projection of a curve on a given surface, the method is to find the scalar $a$ along $P - N$ that lies a point on the surface. In this case, the surface is a sphere, and for each point on curve, the $a$ is found by:

$$a = \frac{2(N-O)^T(N-P')}{(N-P')^2}$$

where $O$ is the center of the sphere, and $P'$ is the position of the point. The position of $P$ on sphere is thus found by

$$P = a(P' - N) + N$$

(b) To represent the orientation in the specified form:

(1) Unit Quaternion

$$q = \begin{bmatrix} cos\frac{\theta}{2} \\ (\sin\frac{\theta}{2})\, k \end{bmatrix}$$

$$cos\frac{\theta}{2} = \sqrt{\frac{1+cos\theta}{2}}$$

$$sin\frac{\theta}{2} = \sqrt{\frac{1-cos\theta}{2}}$$

**Find $\theta$ and $k$ by the method shown in $\beta = k\theta$ below.**

(2) yaw-pitch-roll ZYX
As discussed in lecture, find by:

$$R = \mathsf{R}_z(\beta_1)\mathsf{R}_y(\beta_2)\mathsf{R}_\mathsf{x}(\beta_3).$$
$$\text{find } \beta 2 \text{ by : } e_x^T R e_z = e_x^T R_y(\beta_2) e_z \text{ (Subproblem 4)}$$
$$\text{find } \beta 1 \text{ by : } R e_x(-\beta_1) R e_z = R_y(\beta_2) e_z \text{ (Subproblem 1)}$$
$$\text{find } \beta 3 \text{ by : } R e_z(\beta_3) R^T R e_x = R_y(-\beta_2) e_x \text{ (Subproblem 1)}$$

(3) Angle-axis product
Find by:

$$sin\theta = \pm\frac{||R - R^T||}{2}, cos\theta = \frac{Trace(R) - 1}{2}$$
$$\theta = atan2(sin\theta, cos\theta)$$
$$k = \frac{(R - R^T)^\vee}{2sin\theta}$$

**Results based on simulation:**

(a) The projected curve is shown in figure 1. To match with the view in the example code, the center of the sphere was placed at $O = (0, 0, 2)$, and projected by $N = (-2, 0, 2)$. The original S letter curve is moved from $xy$ plane to $yz$ plane. The code for the stereo projection is `curve_proj.m` in the project folder.

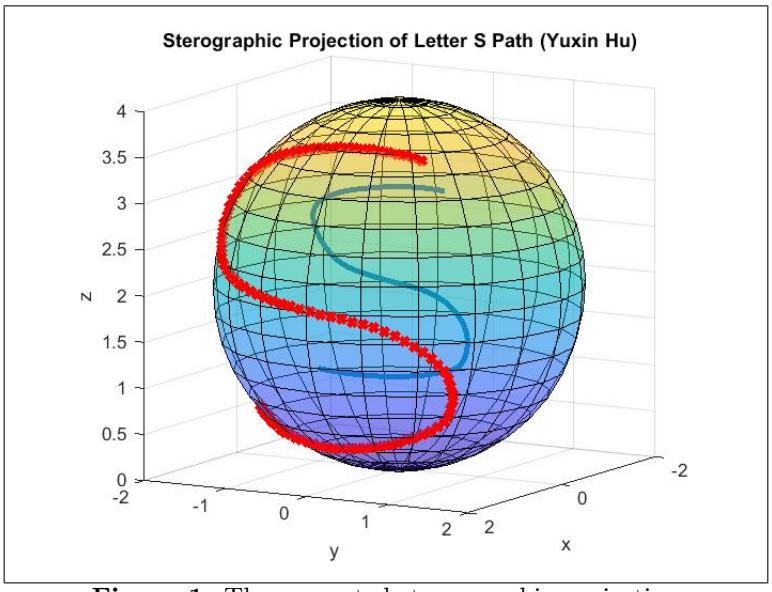


**Figure 1:** The generated stereographic projection

(b) First the curve is modified to make sure the distance between adjacent points on curve to be equal.

(1)The same as the one in the example code, the unit quaternion is displayed by the `plotTransforms` function in matlab, that shows a frame at each point. The result is shown in figure 2.

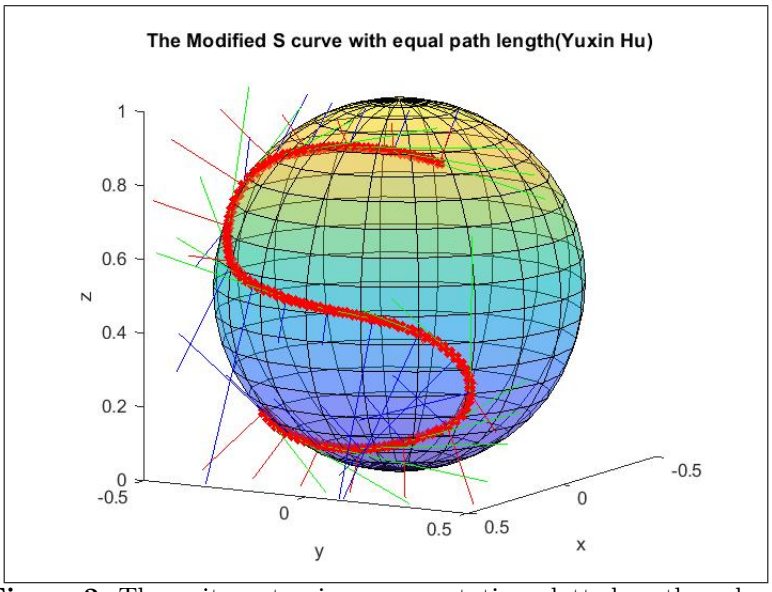


**Figure 2**: The unit quaternion representation plotted on the sphere

Due to the limit of the scope and other factors, the orientation vector is unclear on the plot, thus another view is provided in figure 3, to help better observe the orientation. It is clear that the red lines are perpendicular to the sphere surface, proving that the algorithm is correct.

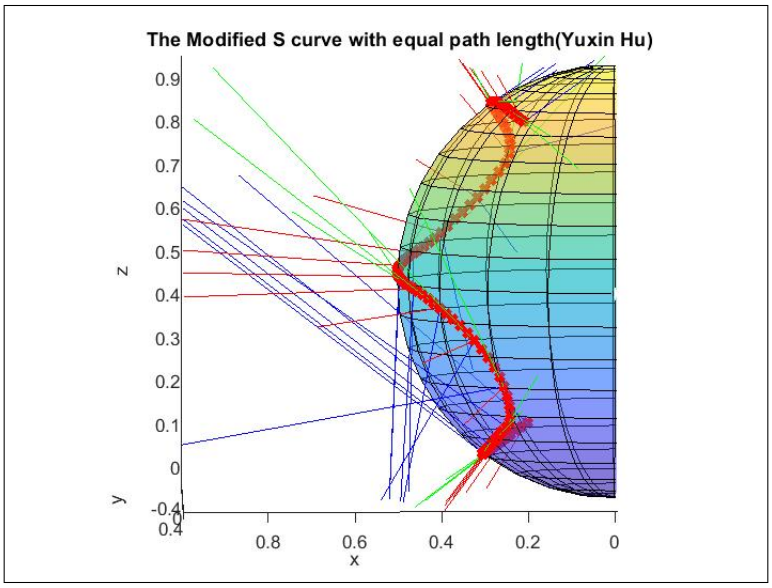

**Figure 3**: Another view of the unit quaternion plotted

(2) The result of the Euler Angles are plotted separately, versus lambda, the path length.

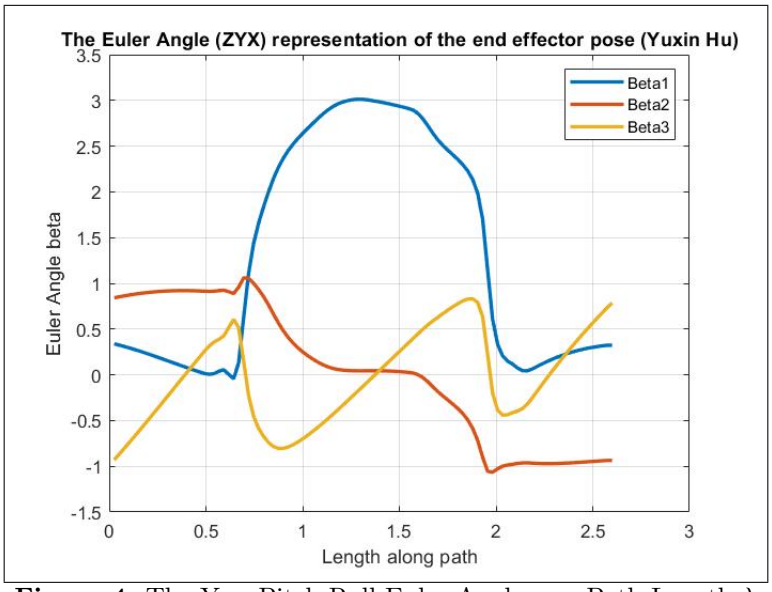


**Figure 4**: The Yaw-Pitch-Roll Euler Angles vs. Path Length $\lambda$

(3) The result of the Axis-Angle Product $k$ and $\theta$ are plotted separately, versus lambda, the path length.

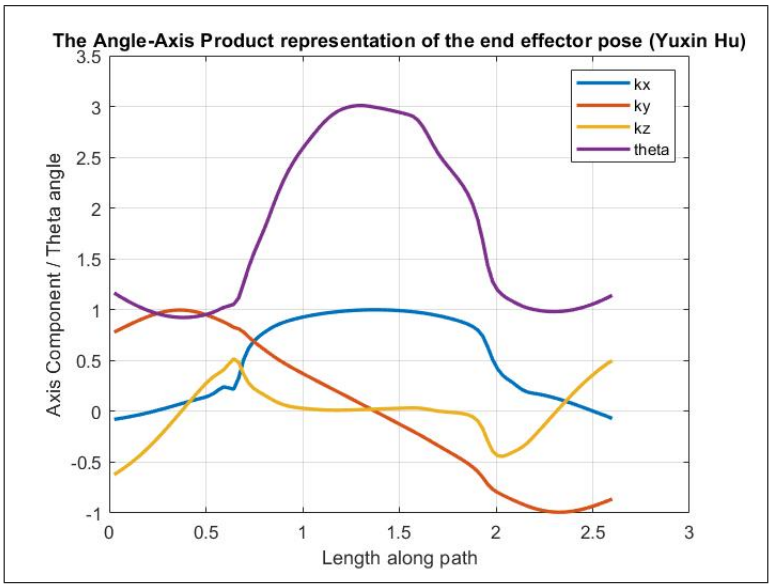


**Figure 5**: The x,y,z component of $k$ and $\theta$ vs. Path Length $\lambda$

## Part 2

**Description of the problem:**

(a) Find the POE, SDH, MDH parameters of the robot

(b) Write forward kinematics code, and verify with SDH and MDH.

(c) Use subproblem decomposition method to solve inverse kinematics.

**Derivation of the solution:**

(a) The Product of Exponential could be used to solve the forward kinematics. In this mini-project, where all joints are rotary, the POE:

$$R_{i-1,i} = R(h_i, q_i) = e^{q_i \cdot h_i^\times}$$
$$R_{0,T} = R_{0,1} \cdot R_{1,2} \cdot ... \cdot R_{n-1,n} \cdot R_{n,T}$$
$$p_{0,T} = p_{0,1} + R_{0,1} \cdot p_{1,2} + R_{0,1}R_{1,2} \cdot p_{2,3} + ... + R_{0,n-1}p_{n-1,n} + R_{0,n}p_{n,T}$$

This is relatively simple since it was covered in lectures for 2D then extended to 3D, so I am not going to discuss into details (Also it's 6:06 AM and I am tired).

The SDH and MDH are found started by sketching a simple diagram of the robot, with only links and joints. Starting from the root, the joints are found by locating either the intersection of $\vec{z_i}$ and $\vec{z_{i-1}}$, or the $O_i$ that would make the distance $d_i$ minimum. Then the $\vec{x_i}$ should be orthogonal to both $\vec{z_i}$ and $\vec{z_{i-1}}$. Once the local frame is built, the SDH parameters could be read by:

$$\vec{p_{i-1,i}} = d_i\vec{z_{i-1}} + a_i\vec{x_i}$$
$$\vec{z_i} = R(\vec{x_i}, \alpha_i)\vec{z_{i-1}}$$
$$\vec{x_i} = R(\vec{z_{i-1}}, \theta_i)\vec{x_{i-1}}$$

and MDH by:

$$\vec{p_{i-1,i}} = d_i\vec{z_i} + a_{i-1}\vec{x_{i-1}}$$
$$\vec{z_i} = R(\vec{x_{i-1}}, \alpha_i)\vec{z_{i-1}}$$
$$\vec{x_i} = R(\vec{z_i}, \theta_i)\vec{x_{i-1}}$$

(b) The POE of a general open-chain robot is shown in (a). To verify the correctness of the POE, SDH, and MDH implemented, perform symbolic calculation in Matlab and see if the output agrees with each other. Since the sample program files was provided, and my program was largely based on them, I guess there is no essence of explaining how it works.

(c) By forward kinematics, it is easy to find:

$$R_{0T} = R_z(q_1) \cdot R_y(q_2) \cdot R_y(q_3) \cdot R_x(q_4) \cdot R_y(q_5) \cdot R_x(q_6)$$
$$p_{0T} = p_{01} + \cancel{R_{01}p_{12}} + R_{01}R_{12}p_{23} + R_{01}R_{12}R_{23}p_{34}$$
$$+\cancel{R_{01}R_{12}R_{23}R_{34}p_{45}} + \cancel{R_{01}R_{12}R_{23}R_{34}R_{45}p_{56}}$$
$$+R_{01}R_{12}R_{23}R_{34}R_{45}R_{56}p_{6T}$$

Move constants to left side:

$$p_{0T} - p_{01} - R_{01}R_{12}R_{23}R_{34}R_{45}R_{56}p_{6T} = R_{01}R_{12}p_{23} + R_{01}R_{12}R_{23}p_{34}$$
$$p_{0T} - p_{01} - R_{0T}p_{6T} = R_{01}R_{12}(p_{23} + R_{23}p_{34})$$
$$||p_{0T} - p_{01} - R_{0T}p_{6T}|| = ||p_{23} + R_{23}p_{34}||$$

Solve the equation for $q_3$ with Subproblem 3 algorithm. Once $q_3$ (2 possible values) are found, solve

$$p_{0T} - p_{01} - R_{0T}p_{6T} = R_{01}R_{12}(p_{23} + R_{23}p_{34})$$

For each q3, solve for $q_1$ and $q_2$, with Subproblem 2 algorithm. $(2 * 2 = 4$ possible sets of values)
Then for each set of $(q_1, q_2, q_3)$, solve

$$R_{0T} = R_z(q_1) \cdot R_y(q_2) \cdot R_y(q_3) \cdot R_x(q_4) \cdot R_y(q_5) \cdot R_x(q_6)$$
$$R_y(-q_2 - q_3) \cdot R_z(-q_1) \cdot R_{0T} \cdot e_x = R_x(q_4) \cdot R_y(q_5) \cdot e_x$$

Solve for $q_4$ and $q_5$ for each set of $(q_1, q_2, q_3)$. $(2 * 2 * 2 = 8$ solutions)
And finally for each set of $(q_1, q_2, q_3, q_4, q_5)$, solve for a unique $q_6$, using Subproblem 1 algorithm.

**Results based on simulation:**

(a) The SDH parameters of the IRB1200 robot is shown below ($R_{6T}$ and $p_{6T}$ not included)

| $i$ | $d_i$ | $a_i$ | $\alpha_i$ | $\theta_i$ |
|---|---|---|---|---|
| 1 | $\ell_1$ | 0 | $-\pi/2$ | $q_1$ |
| 2 | 0 | $\ell_2$ | 0 | $q_2 - \pi/2$ |
| 3 | 0 | -$\ell_3$ | -$\pi/2$ | $q_3$ |
| 4 | $\ell_4$ | 0 | $\pi/2$ | $q_4$ |
| 5 | 0 | 0 | -$\pi/2$ | $q_5$ |
| 6 | 0 | 0 | 0 | $q_6$ |

The MDH parameters of the IRB1200 robot is shown below ($R_{6T}$ and $p_{6T}$ not included)

| $i$ | $d_i$ | $a_{i-1}$ | $\alpha_i$ | $\theta_i$ |
|---|---|---|---|---|
| 1 | $\ell_1$ | 0 | 0 | $q_1$ |
| 2 | 0 | 0 | -$\pi/2$ | $q_2 - \pi/2$ |
| 3 | 0 | -$\ell_2$ | 0 | $q_3$ |
| 4 | $\ell_4$ | $\ell_3$ | -$\pi/2$ | $q_4$ |
| 5 | 0 | 0 | $\pi/2$ | $q_5$ |
| 6 | 0 | 0 | -$\pi/2$ | $q_6$ |

(b) Using the example code file `elbow_POE_SDH_MDH.m`, the result is proved to agree with the POE results. (The original matrix is toooooo large, so only difference result is shown here. Please find `elbow_POE_SDH_MDH.m` in the project folder if you want to learn more about implementation.)

```
difference between POE and SDH forward kinematics

ans =

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

**Figure 6:** The difference between POE and SDH

```
difference between POE and MDH forward kinematics

ans =

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

**Figure 7:** The difference between POE and MDH

(c) To validate the inverse kinematics function, find `verify_invkin.m` in the project folder. As instructed, it generates 6 random $q$ for the robot, and find $R$ via forward kinematics. Then solve the $q$ with the inverse kinematics function, and verify if the original values are in one of the solution sets.

```
q =

   -0.1031
   -1.2795
    2.4527
    0.7977
    1.4736
    1.4745


ans =

    3.0385    3.0385   -0.1031   -0.1031    3.0385    3.0385   -0.1031   -0.1031
   -2.6248   -2.6248   -1.2795   -1.2795    1.2795    1.2795    2.6248    2.6248
    2.4527    2.4527    2.4527    2.4527    0.8746    0.8746    0.8746    0.8746
   -1.0009    2.1407    0.7977   -2.3439   -2.3482    0.7934    2.2257   -0.9159
    1.0087   -1.0087    1.4736   -1.4736    1.6038   -1.6038    1.1157   -1.1157
   -0.8742    2.2674    1.4745   -1.6671    1.6071   -1.5345   -1.0482    2.0934

Forward kinematics and inverse kinematics are working correctly!
```

**Figure 8:** The original $q$ and inverse kinematics solutions

The result above shows that the functions are proved to be working correctly (Notice that the third solution set matches with the original one).

## Part 3

**Description of the problem:**

Use one of the poses to track the S letter curve.

Analyze the speed and variation of joints.

**Derivation of the solution:**

The pose with smallest max angular speed could achieve the highest overall speed when a joint angular speed limit is added. To evaluate the speed of a pose, find tracking speed and tracking time by

$$v = \frac{\dot{q}}{\dot{q}_{max}} = \frac{rad/frame}{rad/s} = frame/s$$

$$t = \frac{f}{v} = \frac{frame}{frame/s} = s$$

In Matlab, there is no concept of time between frames, so using these equations, we could convert the speed into frame per second, and then into running time.

**Results based on simulation:**

One set of solution is plotted in figure 9. The end effector is observed to be tracking the curve perfectly, and the end effector kept perpendicular to the sphere surface. An animation video clip `IRB-1200_invkin.avi` is included in the project folder.

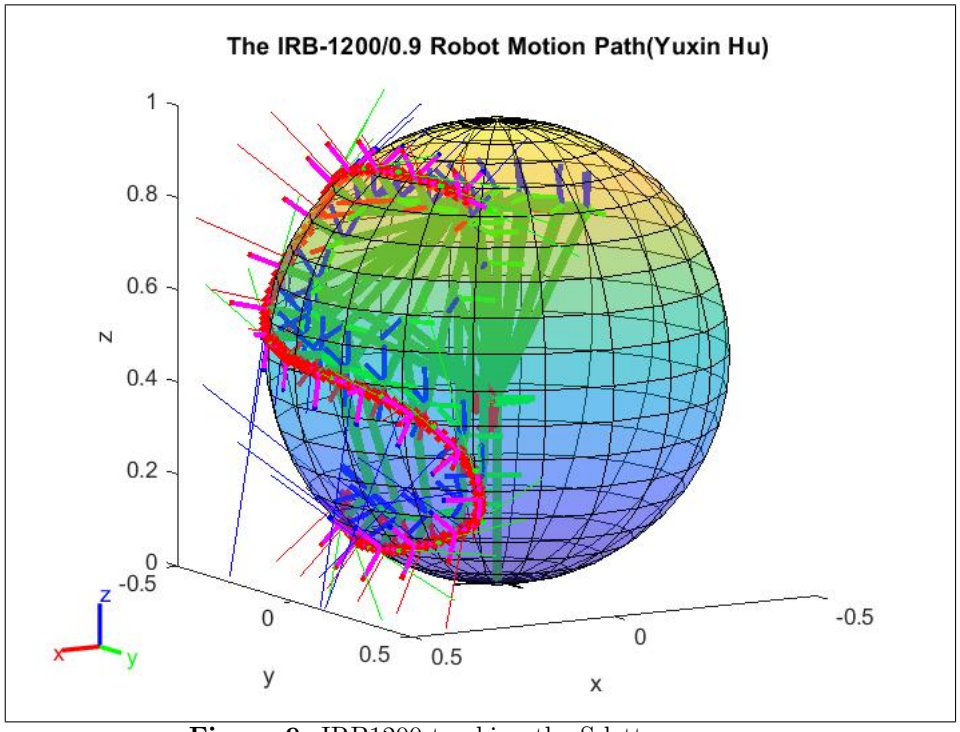


**Figure 9:** IRB1200 tracking the S letter curve

$\lambda$, or the path length, is constant between points, so it will be easy to calculate the highest possible speed. The max angular speed, or $\dot{q}$, is shown in the figure below.

```
maximum qdot for pose 1
    0.0392    2.6452    0.0053         0    0.0026    0.1301

maximum qdot for pose 2
    0.0392    2.6452    0.0053    2.7570    0.0054    2.6200

maximum qdot for pose 3
    2.0003    0.0217    0.0053         0    0.0028    2.6200

maximum qdot for pose 4
    2.0003    0.0217    0.0053    3.0898    0.0045    0.1301

maximum qdot for pose 5
    0.0392    0.0118    0.0023    3.0898    0.0028    2.6200

maximum qdot for pose 6
    0.0392    0.0118    0.0023         0    0.0045    0.1301

maximum qdot for pose 7
    2.0003    0.0272    0.0023    3.0107    0.0026    0.1301

maximum qdot for pose 8
    2.0003    0.0272    0.0023         0    0.0054    2.6200
```

**Figure 10:** The max angular speed of each joint

It is obvious that *pose*6 stands out in all solutions, for the smallest $\dot{q} = 0.0392$. Smaller $\dot{q}$ brings smaller variation. And when the angular speed of each joint is bounded, the one with smallest $\dot{q}$ has the highest possible speed.

$$v = \frac{\dot{q}}{\dot{q}_{max}} = \frac{2rad/s}{0.0392rad/frame} = 51.02frame/s$$
$$t = \frac{f}{v} = \frac{101frame}{51.02frame/s} = 1.9796s$$

The pose 6 could finish the tracking in 1.9796s.