IVR Coursework 1 Report

Hantao Zhong(s1904845), Zhijun Pan(s1969962)

December 3, 2021

1 Work Separation

In this coursework, Hantao worked on the Vision part (2.1 and 2.2) while Zhijun worked on the Control part (3.1 and 3.2). During the process, we had discussions on each other's parts and envisioned ideas and solutions together. GitHub: https://github.com/MrTooOldDriver/UoE-IVR-Coursework.git

2 Robot Vision

2.1 Joint state estimation - I

2.1.1 Algorithm Description

The algorithm consists of two steps. The first is to use two images from camera1 and camera2 to estimate all sphere coordinates in 3D space using blob detection. The second is using geometry vector info to find joint angles.

The blob detection algorithm interprets each image's colour sphere's coordinate. We first use the OpenCV inRange function for each colour of joints to select the sphere colour region and load it as a mask. Then using dilate function to expand and smooth the mask and, after that, use the mask's moment to find the middle point of our mask as our coordinate of a sphere.

Then we could combine spheres coordinates from camera1 and camera2 to reconstruct the 3D coordinate of each sphere. The 3D coordinate $[X, Y, Z]$ where X = - X offset between reference sphere and target sphere in camera2, Y= - X offset between reference sphere and target sphere in camera1 and Z = average Y offset between reference sphere and target sphere in both cameras. In practice, we use the yellow sphere as our reference sphere because the resulting coordinate will be the vector from the yellow sphere. This could reduce error in practice when using the green sphere as the centre of the world since we use lots of yellow sphere vectors for calculation.

After obtaining every sphere's 3D coordinate, we could obtain the vector $\vec{YellowBlue}$. We could find out joint2 by calculating the angle between the z-axis($[0, 0, 1]$) and vector $\vec{YellowBlue}$ projection on the xz plane. ($Vector = [x, y, z]$ projection on xz plane = $[x, 0, z]$). After obtaining joint2 we could calculate the new xz plane projection after joint2 rotations (new projection on xz plane = $[sin(joint2), 0, cos(joint2)]$). Then calculate the angle between the new projection plane and the vector $\vec{YellowBlue}$, we get joint3 angle.

For finding joint4, we first find the vector $\vec{YellowRed}$, then $\vec{YellowRed} - \vec{YellowBlue} = \vec{RedBlue}$. By calculating the angle between $\vec{RedBlue}$ and $\vec{YellowBlue}$, we get joint4. Since the blue joint only could move around one axis.

If any mask moment M['m00'] is zero, then some spheres are not visible. Hence the algorithm will use other spheres locations as no visible sphere location. (e.g. when the blue sphere is invisible, it will use yellow sphere location since it is the only possible situation)

2.1.2 Plots

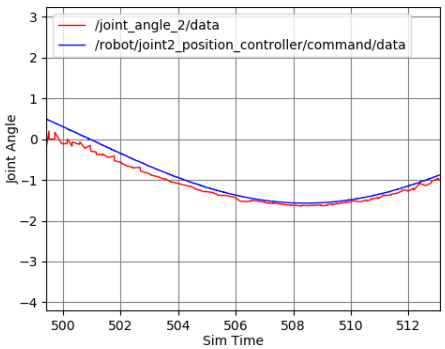

Figure 1: Joint2 result

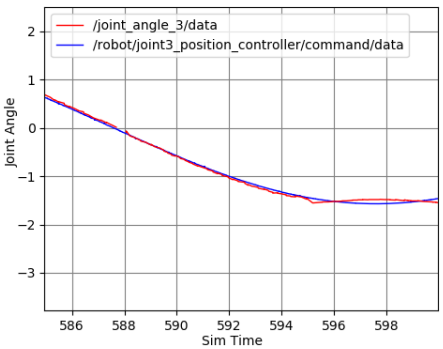

Figure 2: Joint3 result

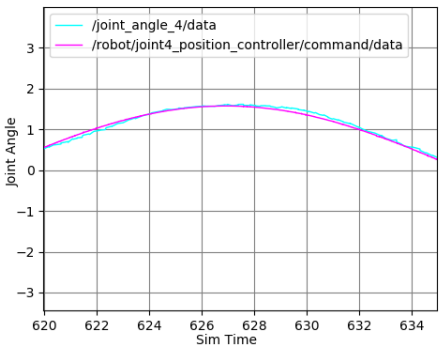

Figure 3: Joint4 result

2.1.3 Observations and Analysis

For joint2, sometimes the estimated value suddenly moves up or down with a huge value(pi or -pi). This is because the sphere partially blocks another sphere, and the sphere detection accuracy will decrease as the visible area reduces as it can't see the whole sphere area. Hence using a moment to estimate visible area will start to become inaccurate.

For joint3, the estimated value is very close to the actual value for all the time. But since we use joint2 to calculate the xz plane projection, our joint2 is sometimes inaccurate, which will affect our joint3 result. This is why we see some peaks and bottom in our result.

For joint4, our algorithm could detect the angle of joint4 very well but can't determine the signed angle due to the nature of calculating the vector angle. It is hard to decide on the angle sign because joint2 and joint3 are constantly changing. The change of joint2 and joint3 will change the reference frame of joint4. Hence, we can't determine whether joint4 moving is positive or negative.

2.2 Joint state estimation - II

2.2.1 Algorithm Description

For this section, the robot only has joint1 3 4 is movable. First, we still use the same method in 2.1 to find each sphere's coordinate in 3D space via two camera images. Then using coordinate to find geometry information of each joint(e.g. vector from yellow sphere to a blue sphere $\vec{YellowBlue}$). For joint1, we calculate and use the angle between the positive y-axis and vector $\vec{YellowBlue}$ as our joint1 angle and the angle between the positive z-axis and vector $\vec{YellowBlue}$ as our joint2 angle. We use vector $\vec{YellowBlue}$ to calculate joint1 is because the movement around the z-axis is not directly visible by our camera. We have to interpret vector $\vec{YellowBlue}$'s direction to blue corresponding with z-axis. But this introduces an issue since joint3 is also moving between $\pi/2$ and $-\pi/2$. The angle between the positive y-axis and vector $\vec{YellowBlue}$ is not the same when joint3 moves in positive angle and negative angle. Hence we can't ensure which angle is the actual angle if we don't know about joint3. However, joint3 also has similar issues since we need to joint1 angle to determine whatever joint3 is positive or negative. Hence the signs of joint1 and joint3 depend on each other. Just like a chicken or the egg question. Therefore we can't always determine the true angles of joint1 and joint3.

For joint4, we still use the same algorithm as 2.1, using the vector angle formula to calculate the angle between vector $\vec{YellowBlue}$ and vector $\vec{BlueRed}$. The signed angle problem remains in the algorithm.

2.2.2 Plots

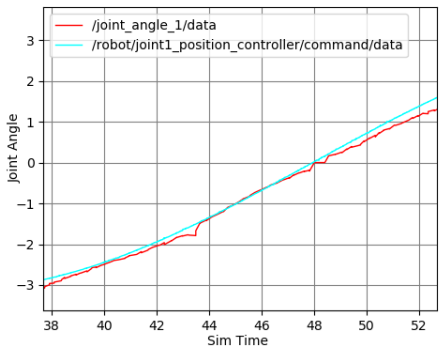

Figure 4: Joint1 result

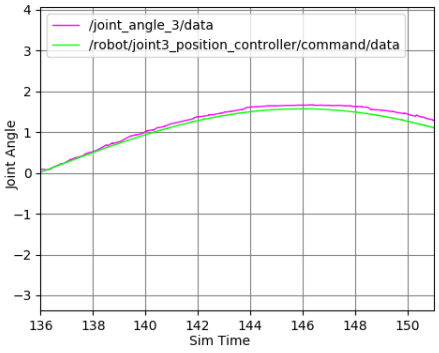

Figure 5: Joint3 result

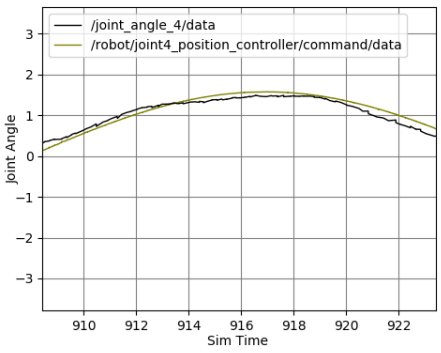

Figure 6: Joint4 result

3 Robot Control

3.1 Forward Kinematics

3.1.1 The Homogeneous Transformation

The homogeneous transformation matrix is:

$$H = \begin{bmatrix} R_n^0 & O_n^0 \\ 0 & 1 \end{bmatrix}, A_i = \begin{bmatrix} R_i^{i-1} & O_i^{i-1} \\ 0 & 1 \end{bmatrix} \tag{1}$$

where R is a 3x3 matrix representing rotation and O is a 3d vector representing coordinates which in this represents the displacements (translations) between joints. The final transformation matrix H is a multiplication of transformation matrices A on each joint angle where $H = A_1(q_1)...A_n(q_n)$. Hence, for the transformation from the origin to joint 1, 3, 4 and the end-effector, the final matrix H is (the result of H is way too long so we omitted it here):

$$H = A_1A_3A_4A_{end} = \begin{pmatrix} cos\,(\theta_1) & -sin\,(\theta_1) & 0 & 0 \\ sin\,(\theta_1) & cos\,(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\,(\theta_3) & -sin\,(\theta_3) & 0 \\ 0 & sin\,(\theta_3) & cos\,(\theta_3) & 4 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos\,(\theta_4) & 0 & sin\,(\theta_4) & 0 \\ 0 & 1 & 0 & 0 \\ -sin\,(\theta_4) & 0 & cos\,(\theta_4) & 3.2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2.8 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

To only get the last column of H (which is $\begin{pmatrix} O_n^0 \\ 1 \end{pmatrix}$), we multiply H by the vector $(0\,0\,0\,1)$:

$$\begin{pmatrix} O_n^0 \\ 1 \end{pmatrix} = H \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 3.2\sin(\theta_1)\sin(\theta_3) + 2.8\,(\cos(\theta_1)\sin(\theta_4) + \sin(\theta_1)\sin(\theta_3)\cos(\theta_4)) \\ 2.8\,(\sin(\theta_1)\sin(\theta_4) - \cos(\theta_1)\sin(\theta_3)\cos(\theta_4)) - 3.2\cos(\theta_1)\sin(\theta_3) \\ 2.8\cos(\theta_3)\cos(\theta_4) + 3.2\cos(\theta_3) + 4 \\ 1 \end{pmatrix} \tag{3}$$

where O_n^0 are the end-effector's position coordinates.

3.1.2 The Result of Forward Kinematics Calculation

The calculation of homogeneous transformation matrix H is shown in the last section, and the 3-d vector O_n^0 is the final location of the end-effector. Hence, the forward kinematics in this case is:

$$k_e = K(\mathbf{q}) = O_n^0 = \begin{pmatrix} 3.2\sin(\theta_1)\sin(\theta_3) + 2.8\,(\cos(\theta_1)\sin(\theta_4) + \sin(\theta_1)\sin(\theta_3)\cos(\theta_4)) \\ 2.8\,(\sin(\theta_1)\sin(\theta_4) - \cos(\theta_1)\sin(\theta_3)\cos(\theta_4)) - 3.2\cos(\theta_1)\sin(\theta_3) \\ 2.8\cos(\theta_3)\cos(\theta_4) + 3.2\cos(\theta_3) + 4 \end{pmatrix} \tag{4}$$

where k_e is the coordinates of the end-effector and $\mathbf{q} = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{pmatrix}$.

3.1.3 Verifying the Results of Forward Kinematics Calculation

A table comparing the end-effector positions via FK calculation to detection using images:

\	Joint angles [θ_1, θ_3, θ_4]	FK calculation result $[x, y, z]$	Detection from image $[x, y, z]$
1	[1.0, 1.0, 1.0]	[4.610, -0.160, 6.546]	[4.650, -0.035, 5.981]
2	[1.3, 1.5, 0.8]	[5.487, 0.561, 4.364]	[5.183, 0.852, 3.603]
3	[1.6, 0.8, 2.0]	[1.384, 2.587, 5.417]	[1.846, 3.195, 5.626]
4	[2.5, 1.1, 0.4]	[2.208, 4.778, 6.621]	[1.881, 5.254, 5.999]
5	[1.1, 2.0, 1.2]	[4.599, 0.587, 2.246]	[4.828, 0.816, 3.053]
6	[1.9, 0.5, 0.6]	[1.989, 2.350, 8.836]	[1.881, 2.520, 8.182]
7	[3.0, 2.8, 2.5]	[-1.613, 0.553, 3.098]	[-1.775, -0.071, 3.851]
8	[2.8, 2.0, 1.0]	[-0.784, 4.827, 2.038]	[-0.461, 4.828, 3.709]
9	[0.8, 0.8, 0.8]	[4.049, -1.133, 7.588]	[4.366, -1.242, 7.188]
10	[0.6, 0.7, 0.9]	[3.607, -1.388, 7.778]	[3.976, -1.526, 7.401]

Comments: From the table, we can see that the end-effector positions calculated via FK is always close to the positions via detection using images. As the result of FK calculation is definite, the difference only comes the errors in detection. By the data in the table plus verification by looking at the output images, the FK calculations can be confirmed correct and accurate.

3.2 Inverse Kinematics

3.2.1 Calculating Joint Angles

In order to get the end-effector to a target position, we calculate the joint angles required to achieve the effect. We have $\dot{\mathbf{x}} = J(\mathbf{q})\dot{\mathbf{q}}$ where $J(\mathbf{q})$ is the Jacobian matrix, $\dot{\mathbf{x}}$ is the target position, $\mathbf{q}$ is the current joint angles and $\dot{\mathbf{q}}$ is the desired joint angles. Hence $\dot{\mathbf{q}} = J^{-1}\dot{\mathbf{x}}$ where J^{-1} is the inverse of Jacobian matrix.

3.2.2 Calculating Jacobian and Its Inverse

Jacobian matrix is a partial derivative of the FK equation $K(\mathbf{q})$ with respect to change of time dt. So (the resultant J is too long so separated into columns):

$$
\begin{aligned}
J = \frac{\partial K(\mathbf{q})}{\partial \mathbf{q}} = \\
column\ 0: & \begin{pmatrix} -2.8\sin(\theta_1)\sin(\theta_4) + 2.8\sin(\theta_3)\cos(\theta_1)\cos(\theta_4) + 3.2\sin(\theta_3)\cos(\theta_1) \\ 2.8\sin(\theta_1)\sin(\theta_3)\cos(\theta_4) + 3.2\sin(\theta_1)\sin(\theta_3) + 2.8\sin(\theta_4)\cos(\theta_1) \\ 0 \end{pmatrix} \\
column\ 1: & \begin{pmatrix} 2.8\sin(\theta_1)\cos(\theta_3)\cos(\theta_4) + 3.2\sin(\theta_1)\cos(\theta_3) \\ -2.8\cos(\theta_1)\cos(\theta_3)\cos(\theta_4) - 3.2\cos(\theta_1)\cos(\theta_3) \\ -2.8\sin(\theta_3)\cos(\theta_4) - 3.2\sin(\theta_3) \end{pmatrix} \\
column\ 2: & \begin{pmatrix} -2.8\sin(\theta_1)\sin(\theta_3)\sin(\theta_4) + 2.8\cos(\theta_1)\cos(\theta_4) \\ 2.8\sin(\theta_1)\cos(\theta_4) + 2.8\sin(\theta_3)\sin(\theta_4)\cos(\theta_1) \\ -2.8\sin(\theta_4)\cos(\theta_3) \end{pmatrix}
\end{aligned}
\tag{5}
$$

As the Jacobian is not inevitable in general, we calculate its pseudo-inverse: $J^{\dagger} = J^T(JJ^T)^{-1}$.

3.2.3 Executing Inverse Kinematics

Once we have the inverse of Jacobian matrix, we can calculate the desired $q_{des} = q_{curr} + \Delta t J^{-1}\dot{\mathbf{x}}$ where Δt is the time difference since last execution and q_{curr} are the current joint angles' states. After iterations, the end-effector's position should approach the target position.

3.2.4 Plots comparing the x, y, and z positions

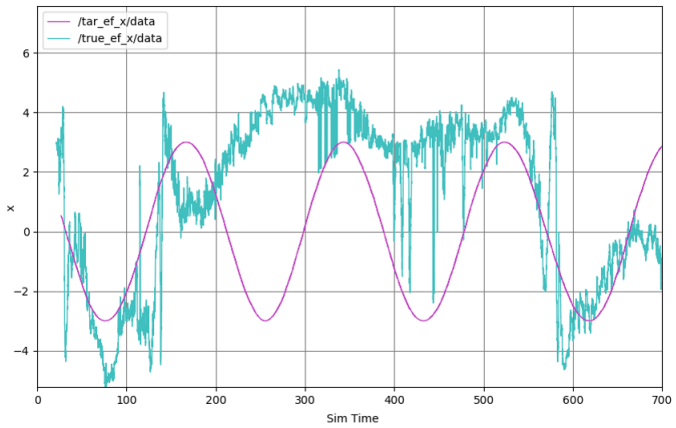

Figure 7: end-effector target vs publish on x

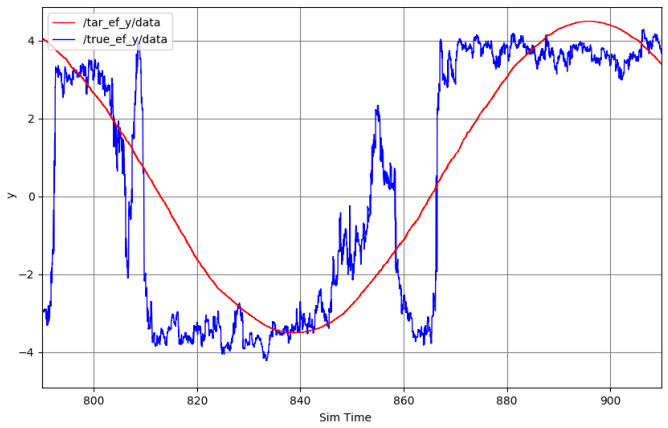

Figure 8: end-effector target vs publish on y

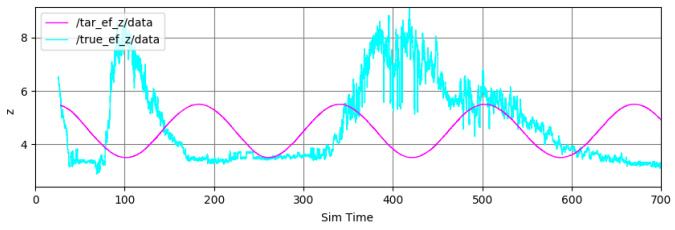

Figure 9: end-effector target vs publish on z