IVR Assignment Report

Srihari Humbarwadi and Thanakrit Anutrakulchai

December 2, 2021

1 Contributions

For the vision part (both files), Thanakrit worked on deriving formulae for joint angles from detected blob positions. Srihari worked on image normalization, background masking for blob detection, correcting for bias, and helped simplify the calculations above. For the Forward Kinematics, Inverse Kinematics part, We both calculated the forward kinematics and Jacobian matrices by hand. Thanakrit populated the D-H table, as well as the table comparing measured x, y, z positions to those derived from Forward Kinematics and helped with the gradient calculation. Srihari worked on designing the optimization and error function for Inverse kinematics solution and helped computing and reporting error in Forward Kinematics.
repository: `https://github.com/srihari-humbarwadi/ivr_assignment`

2 Notation

For this report, we use ϑ, ν, ϕ, ψ to denote the angles in radians of joints 1, 2, 3, and 4, respectively. We denote the position of the i^{th} joint with x_i, y_i, z_i, and x_e, y_e, z_e for the position of the end-effector, from any arbitrary left-handed co-ordinate frame. (We only use the differences between values so the position of the frame does not matter. We choose a left-hand frame as that made getting positional data from the cameras more straightforward.) We also use L_1, L_2, L_3, L_4 for the lengths of the respective links. We use short-hand notation for sine and cosine, as in s_γ is to be read as $\sin(\gamma)$ and c_γ as $\cos(\gamma)$ for angles γ.

3 Joint State Estimation

3.1 Part I —Fixing Joint 1

To calculate the angles of the joints, we need to localize the joints in 3D space. We first normalize the image using *RGB* normalization, then mask the background and the links between the joint. This ensures that our resultant image only has joint spheres present. We use thresholding to generate binary masks of the green, yellow, blue, and red spheres which correspond to joints 1, 2, 3, 4 and the end-effector respectively. As we have two cameras, one facing the yz-plane, and the other facing the xz-plane, we perform this operation on the images received from both camera, and combine their results. Thus, we use *camera 1* to localize the blobs in *yz* plane and use *camera 2* to localize the blobs in *xz* plane.

We encode the orientations of links 3 and 4 by the unit-length vectors $\mathbf{r}_3(\mathbf{q})$, $\mathbf{r}_4(\mathbf{q})$ pointing from joint 2 to 3, and joint 3 to 4, respectively. We choose two frames so when all joints are at angle 0, the unit vectors have co-ordinates [0 0 1] in those frames. These vectors are rotated as the joint angles change as so:

$$\mathbf{r}_3(\mathbf{q}) := \begin{pmatrix} r_{3x} \\ r_{3y} \\ r_{3z} \end{pmatrix} = \begin{pmatrix} c_\phi s_\nu \\ -s_\phi \\ c_\phi c_\nu \end{pmatrix} \approx \frac{1}{\sqrt{(x_4 - x_3)^2 + (y_4 - y_3)^2 + (z_3 - z_4)^2}} \begin{pmatrix} x_4 - x_3 \\ y_4 - y_3 \\ z_3 - z_4 \end{pmatrix}$$

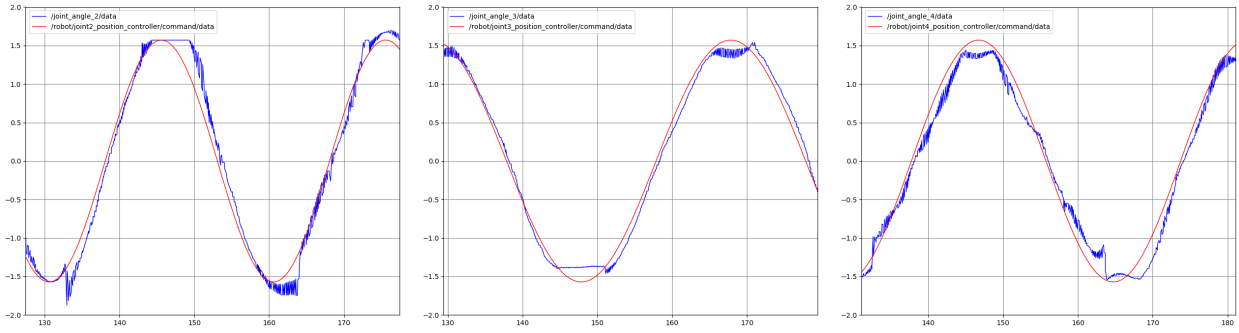
$$\mathbf{r}_4(\mathbf{q}) := \begin{pmatrix} r_{4x} \\ r_{4y} \\ r_{4z} \end{pmatrix} = \begin{pmatrix} c_\nu s_\psi + s_\nu c_\phi c_\psi \\ -s_\phi c_\psi \\ -s_\nu s_\psi + c_\nu c_\phi c_\psi \end{pmatrix} \approx \frac{1}{\sqrt{(x_e - x_4)^2 + (y_e - y_4)^2 + (z_4 - z_e)^2}} \begin{pmatrix} x_e - x_4 \\ y_e - y_4 \\ z_4 - z_e \end{pmatrix}$$

The formulae above can be derived from geometry or multiplying the relevant rotation matrices. We negate the z component as we switch from a left-handed frame to right-handed frames.

With these values, we can calculate the joint angles as so:

$$\nu = \arctan 2(r_{3x}, r_{3z}) \qquad ||\nu = \arctan 2(r_{4x}c_\phi c_\psi - r_{4z}s_\psi,\ \ r_{4x}s_\psi + r_{4z}c_\phi c_\psi)$$
$$\phi = \arcsin(-r_{3y})$$
$$\psi = \arcsin(r_{4x}c_\nu - r_{4z}s_\nu)$$

where we only use the right-hand formula for ν when $\phi \approx \pm\frac{\pi}{2}$, $r_{3x},\ r_{3z} \approx 0$, so $\arctan 2(r_{3x}, r_{3z})$ oscillates rapidly. Additionally, when $\psi \approx 0$ also, we stop calculating ν and keep the angle published constant. As we have a gimbal lock situation, any value of ν produces the same observed positions, so our only information on ν is it cannot physically change from its previous value too much.



Graphs of estimated angles and actual angle (y-axis) vs. time (x-axis). As mentioned earlier, ν stays constant when gimbal lock occurs. Joint 3's estimation undershoots due to small measurement errors —let ϵ_x, ϵ_z be small Gaussian random variables centered at 0 approximating the true value of $r_{3x},\ r_{3z} \approx 0$ when $\phi \approx \pm\frac{\pi}{2}$. Then, r_{3y} is calculated to be $\pm\sqrt{1-\epsilon_x^2-\epsilon_z^2}$, which is noticeably smaller than 1 if the measurement errors are large. The estimated angle of joint 2 overshoots during peaks, while that of joint 4 undershoots, as the two angles are heavily coupled when $\phi \approx \pm\frac{\pi}{2}$. The lines of the estimated angles oscillate, due to using z-coordinate data from both cameras, and not correcting for when the distance from the joints to the cameras change as they move.

3.2 Part II —Fixing Joint 2

We follow the same methodology from Part I, with a slight modification as most observed sets of positions now correspond to two possible sets of angles. We calculate the unit-vectors now as:

$$\mathbf{r}_3(\mathbf{q}) := \begin{pmatrix} r_{3x} \\ r_{3y} \\ r_{3z} \end{pmatrix} = \begin{pmatrix} s_\vartheta s_\phi \\ -c_\vartheta s_\phi \\ c_\phi \end{pmatrix} \approx \frac{1}{\sqrt{(x_4-x_3)^2+(y_4-y_3)^2+(z_3-z_4)^2}} \begin{pmatrix} x_4-x_3 \\ y_4-y_3 \\ z_3-z_4 \end{pmatrix}$$

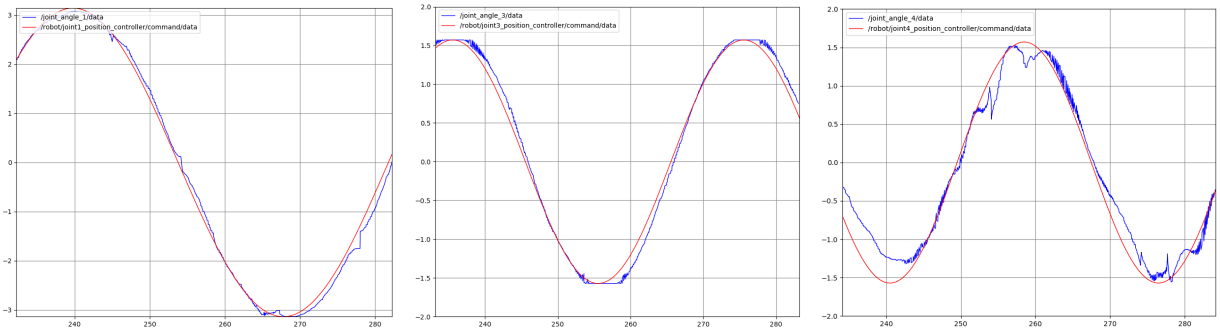
$$\mathbf{r}_4(\mathbf{q}) := \begin{pmatrix} r_{4x} \\ r_{4y} \\ r_{4z} \end{pmatrix} = \begin{pmatrix} c_\vartheta s_\psi + s_\vartheta s_\phi c_\psi \\ s_\vartheta s_\psi - c_\vartheta s_\phi c_\psi \\ c_\phi c_\psi \end{pmatrix} \approx \frac{1}{\sqrt{(x_e-x_4)^2+(y_e-y_4)^2+(z_4-z_e)^2}} \begin{pmatrix} x_e-x_4 \\ y_e-y_4 \\ z_4-z_e \end{pmatrix}$$

and the angles as:

$$\vartheta = \arctan 2(r_{3x}sgn(\phi), -r_{3y}sgn(\phi)) \qquad ||\vartheta = \arctan 2(r_{4x}s_\phi c_\psi + r_{4y}s_\psi,\ \ r_{4x}s_\psi - r_{4y}s_\phi c_\psi)$$
$$\phi = sgn(\phi)\arccos(\phi) \qquad ||\phi = \arcsin(r_{3x}s_\vartheta - r_{3y}c_\vartheta)$$
$$\psi = \arcsin(r_{4x}c_\vartheta + r_{4z}s_\vartheta)$$

where the sign function $sgn(\phi)$ is -1 if $\phi < 0$ in the previous iteration, and 1 otherwise. We only use the formulae on the right-hand side when We have problems with $\arctan 2$ and $sgn(\phi)$ when $\phi,\ r_{3x},\ r_{3y} \approx 0$. Additionally, when ψ is also near zero, we have a gimbal lock situation, so we keep ϑ, and thus also ϕ, constant during these periods.

Earlier, we noted that most sets of positions correspond to two possible sets of angles. We choose a set arbitrarily at the start and ensure the angle values do not jump discontinuously. There is an exception: when ϑ crosses between π and $-\pi$, we realize we started with the wrong set of angles, so we switch to the other set by adding or subtracting π from ϑ as appropriate, and negating the other angles.



Graphs of estimated angles and actual angle (y-axis) vs. time (x-axis)

4 Control

4.1 Forward Kinematics

We derived Forward Kinematics for the robot using the D-H convention. The D-H table we derived is from following the usual procedures. However, whenever the x-axis of a frame will be parallel or anti-parallel to that of the next frame, our convention is to choose the x-axis of the next frame so that they are parallel. (so sometimes we use the cross product $z_t \times z_{t+1}$ and sometimes $z_{t+1} \times z_t$, following this convention).

link	α	a	d	θ
1	$-\frac{\pi}{2}$	0	L_1	ϑ
2	$-\frac{\pi}{2}$	0	0	$-\frac{\pi}{2}$
3	$\frac{\pi}{2}$	L_3	0	ϕ
4	0	L_4	0	ψ

$$K(\mathbf{q}) = \begin{pmatrix} c_\vartheta s_\psi L_4 + s_\vartheta s_\phi c_\psi L_4 + s_\vartheta s_\phi L_3 \\ s_\vartheta s_\psi L_4 - c_\vartheta s_\phi c_\psi L_4 - c_\vartheta s_\phi L_3 \\ c_\phi c_\psi L_4 + c_\phi L_3 + L_1 \end{pmatrix}$$

To calculate $K(\mathbf{q})$ from this table, please follow the exact procedure outlined in the Denavit-Hartenberg.pdf, slides 6 and 9, in the Forward Kinematics part of the Video Lectures section of the IVR course's learn page.

ϑ	ϕ	ψ	x_e	y_e	z_e	K_x	K_y	K_z	MSE
2.3	0.4	-0.2	1.98	1.27	9.60	2.10	1.13	9.47	0.017
-0.7	1.2	0.3	-3.96	-4.09	5.94	-2.89	-4.72	6.13	0.526
1.1	1.1	1.1	4.58	0.69	5.76	4.68	0.42	6.03	0.052
0.5	-0.9	1.2	0.64	4.36	6.20	0.71	4.15	6.62	0.075
-1.8	-0.4	-0.4	2.38	0.68	9.52	2.44	0.55	9.32	0.020
0.6	1.3	-0.7	1.96	-5.76	4.97	1.42	-5.27	5.43	0.248
-2.5	0.5	1.4	-3.38	-0.28	6.46	-3.27	-0.24	7.23	0.202
0.3	0.9	-0.9	-1.35	-4.13	7.14	-0.95	-4.35	7.07	0.071
2.7	-1.2	1.3	-4.63	-1.71	4.97	-4.01	-2.17	5.43	0.269
0.2	0.3	-0.2	-0.24	-1.81	9.20	-0.20	-1.83	9.68	0.077

Table 1: the x, y, z coordinates of the end-effector obtained from forward kinematics K_x, K_y, K_z, compared against the measured positions x_e, y_e, z_e, at different joint angle values.

One of the major contributors to the error is the *pixel to meter* conversion constant, which is directly affected by the distance of the joint links from the cameras. On a average, we get an *mean squared error* of **0.155** for the above configurations.

4.2 Inverse Kinematics

Jacobian calculated from Forward kinematics

$$J(\mathbf{q}) = \begin{pmatrix} -s_\vartheta s_\psi L_4 + c_\vartheta s_\phi c_\psi L_4 + c_\vartheta s_\phi L_3 & s_\vartheta c_\phi c_\psi L_4 + s_\vartheta c_\phi L_3 & c_\vartheta c_\psi L_4 - s_\vartheta s_\phi s_\psi L_4 \\ c_\vartheta s_\psi L_4 + s_\vartheta s_\phi c_\psi L_4 + s_\vartheta s_\phi L_3 & -c_\vartheta c_\phi c_\psi L_4 - c_\vartheta c_\phi L_3 & s_\vartheta c_\psi L_4 + c_\vartheta s_\phi s_\psi L_4 \\ 0 & -s_\phi c_\psi L_4 - s_\phi L_3 & -c_\phi s_\psi L_4 \end{pmatrix}$$

We quantify the discrepancy between the current position of the end-effector and the target using an error function. An ideal solution would be to have a *zero* error, which means that our end-effector is exactly at the required target position. We formulate this as an optimization problem and try to minimize the error function. [1]

Error function: Empirically we found that using mean of squared errors across the three x, y and z coordinates gave us the best performance. Here *target* and *current* are the target coordinates and the current coordinates for the end-effector respectively.

$$E(\textbf{target, current}) = \frac{1}{3} * [(x_t - x_c)^2 + (y_t - y_c)^2 + (z_t - z_c)^2]$$

Optimization Method: We selected a *first-order* iterative optimization algorithm — *Gradient Descent* [2] to minimize our error function. We update our joint configuration $\mathbf{q}$ iteratively using the following update rule

$$\mathbf{q}_{t+1} \longleftarrow \mathbf{q}_t - \alpha * \frac{\partial E(\textbf{target, current})}{\partial \mathbf{q}}$$

Here, α is the step size of the update, we found that using *0.01* yields the best performance. We exit the optimization process when the *mean absolute error* for the coordinates drops below *0.005* or when the number of iterations exceed *500* iterations.

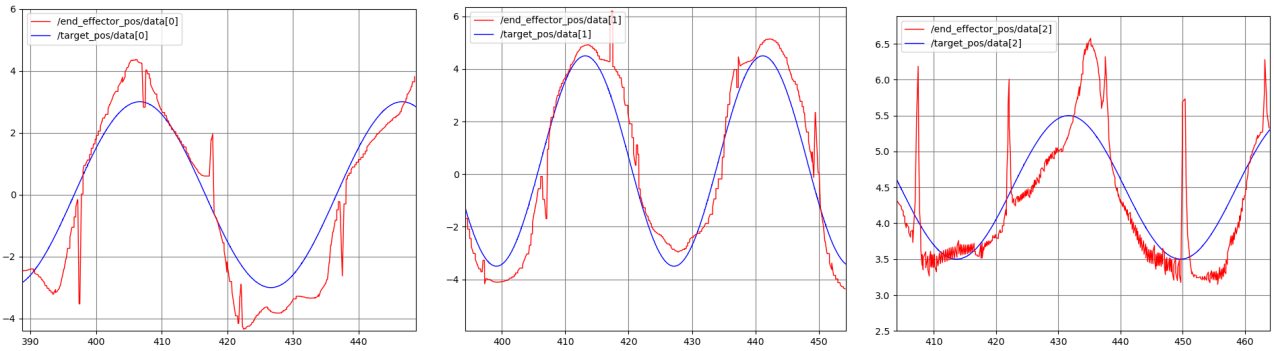
The gradient of the error function given by $\frac{\partial E(\textbf{target, current})}{\partial \mathbf{q}}$ can be expressed in terms of $J(\mathbf{q})$ since the current position of the end-effector is calculated by forward kinematics given by $K(\mathbf{q})$. Hence the gradient can be re-written as $\frac{2}{3} * (\textbf{target} - K(\mathbf{q})) * J(\mathbf{q})$

For each new target received, we have three strategies to initialize our initial joint configuration $\mathbf{q}_0$. We could start from the same initial seed configuration always, or use the previously computed solution for our next target. But, we found that randomly initializing joint configuration such that $\mathbf{q}_0 \sim \mathcal{N}(0, 0.005)$ helps us avoid getting stuck in *local minima(s)*.

Constraints: The individual joints have constraints on the amount of rotation allowed. We make sure that our solutions is bound by these constraints.

Joint Angle Estimations: Since we directly evolve our solution from a random initial configuration for each new target that is published, we **do not** require any external angle estimations. We do not subscribe to any joint angle estimations from *vision 2* for the same reason.

External Package: To make sure that the above mentioned solution runs in real-time, we use an external package — *TensorFlow* [3], which provides primitives for efficiently computing gradients and running the *Gradient Descent* update rule. We only use this package for its speed.



References

[1] Steve Rotenberg. *Inverse Kinematics*. URL: `https://cseweb.ucsd.edu/classes/sp16/cse169-a/slides/CSE169_08.pdf`.

[2] Wikipedia. *Gradient descent*. URL: `https://en.wikipedia.org/wiki/Gradient_descent`.

[3] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.