Introduction to Vision and Robotics Assignment - s1704145 & s1703913

https://github.com/soniammarshall/ivr_assignment

2.1 Joint State Estimation

To calculate the joint angles, we used the library function scipy.optimize.least_squares to find the minimum error between a) the unit vector from the blue blob to the green blob and b) the orientation of the x3 vector in the third frame (obtained by looking at the first three rows within the first column of the homogeneous transformation A1A2A3). The two are meant to be identical in orientation, and by finding the angles $\Theta_1,\Theta_{2,}\Theta_3$ which best approximate them, we know they ought to be the correct angles. To find the last angle Θ_4, we perform inverse rotations on the normalized vector from the green blob to the red blob until we get its orientation in the reference frame. After this, we know that Θ_4 rotates around z in the fourth frame and therefore by calculating the arctangent of the Y and X displacement in the reference frame of the vector x4, we obtain the required angle.

In moments when a joint is partly or fully hidden by the joints, links, or targets this can cause our estimation of the blob positions to be incorrect, and this results in the vectors mentioned above being incorrect, and therefore the joint angle estimate will be off. If we had more time, we would track the history of the blob positions, detect when the full blob is not visible, and in these cases use the blob history to predict a new blob position similar to the history.

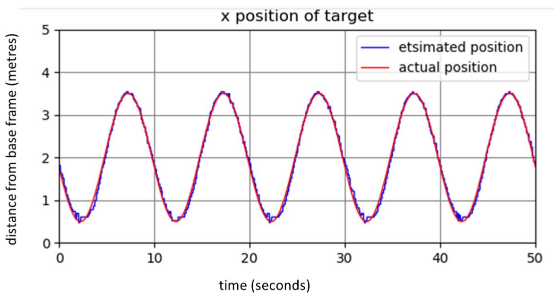
Joint angle command ($\Theta_1,\Theta_{2,}\Theta_{3,}\Theta_4$)	**Estimated joint angle ($\Theta_1,\Theta_{2,}\Theta_{3,}\Theta_4$)**
(0.30, 0.10, 0.50, 0.50)	(0.14, 0.04, 0.59, 1.07)
(0.10, 0.10, 0.10, 0.10)	(0.48, 0.08, 0.18, 0.32)
(0.20, 0.30, 0.10, 0.40)	(0.31, 0.13, 0.32, 0.64)
(0.30, 0.60, 0.10, 0.40)	(0.45, 0.44, 0.55, 1.54)
(0.20, 0.20, 0.60, 0.10)	(0.04, 0.04, 0.72, 0.850)
(0.46, 0.12, 0.67, 0.45)	(0.14, 0.15, 0.77, 1.63)
(0.35, 0.31, 0.54, 0.65)	(0.07, 0.08, 0.72, 1.78)
(0.84, 0.35, 0.73, 0.23)	(0.24, 0.31, 0.83, 2.25)
(0.23, 0.21, 0.32, 0.05)	(0.21, 0.12, 0.46, 0.45)
(0.53, 0.71, 0.44, 0.41)	(0.00, 0.02, 0.99, 2.27)

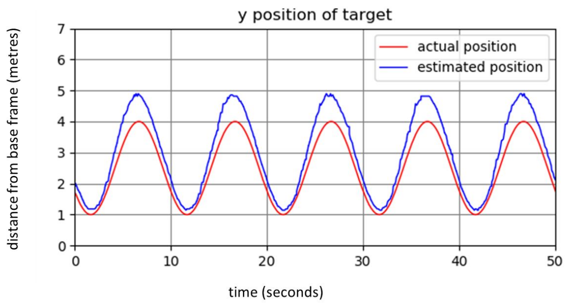
Sources of error between estimates and actual: initial angle position might descend into a local minimum which doesn't give us the correct estimation, our matrix rotation transformations upon which we heavily rely here are taken from our FK equation, which as discussed further below is most probably incorrect.

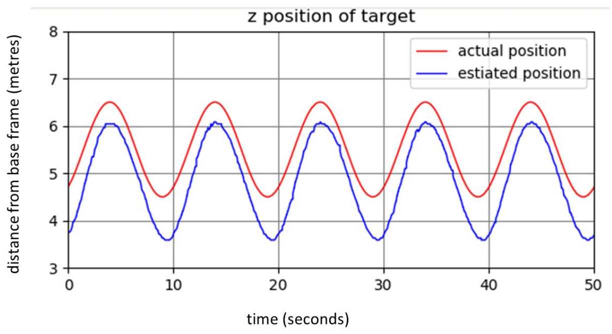
2.2 Target Detection

To perform **target detection**, we used vision algorithms from the CV2 library: cv2.inRange to isolate the orange color, cv2.erode and cv2.dilate to remove noise, cv2.moment to get sphere centroid, cv2.matchTemplate with a snapshot of the sphere as template to get a matrix of possible locations. This matrix returned two great matches (intuitively, one for the box and one for the sphere). The **correct target (sphere)**, however, had a higher match as it fitted the sphere template better. To find the **position with respect to the base frame**, we thresholded the image to find yellow pixels, found the yellow centroid, and subtracted it from the sphere position.

Figures of target estimates vs actual position







Sources of error between estimates and actual: conversion from pixels to meters in camera 1 (ZY), choice of the Z position of the sphere (could be either camera, we tested 1, 2, and average and found the results from camera 1 to be more accurate so we used that), use of the history of the orange sphere in the case when it's hidden in one of the cameras.

3.1 Forward Kinematics

Our calculations done to obtain this result can be found in the LaTeX PDF 'Forward Kinematics', in our GitHub repository. x_e , y_e and z_e are the end effector positions. θ_1 , θ_2 , θ_3 and θ_4 are the joint angles:

$$\begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} = \begin{bmatrix} (-\cos\theta_1\sin\theta_3 + \sin\theta_1\sin\theta_2\cos\theta_3)(2\cos\theta_4+3) + 2\sin\theta_4(-\sin\theta_1\sin\theta_2\sin\theta_3 - \cos\theta_1\cos\theta_3) \\ (-\sin\theta_1\sin\theta_3 - \cos\theta_1\sin\theta_2\cos\theta_3)(2\cos\theta_4+3) + 2\sin\theta_4(-\cos\theta_1\sin\theta_2\sin\theta_3 + \sin\theta_1\cos\theta_3) \\ \cos\theta_2\cos\theta_3(2\cos\theta_4+3) + 2(-\sin\theta_4\cos\theta_2\sin\theta_3 + 1)) \end{bmatrix}$$

Input joint angles (Θ_1,Θ_2,Θ_3,Θ_4)	**Estimated end-effector position (Red Mask Thresholding) (x, y, z)**	**Estimated end-effector position (Forward Kinematics) (x, y, z)**
(0.53, 0.71, 0.44, 0.41)	(4.70, 2.73, 4.69)	(1.07, 3.33, 5.06)
(0.59, 0.55, 0.97, 0.91)	(5.22, 0.33, 2.29)	(3.32, 3.05, 2.93)

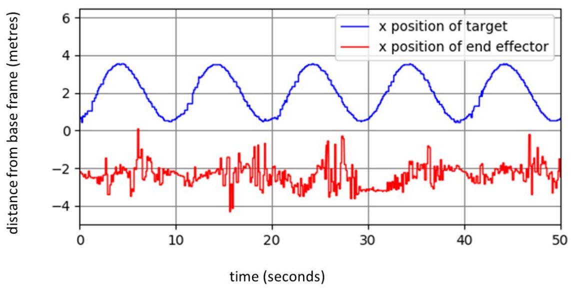
(0.21, 0.35, 0.52, 0.23)	(3.62, 1.75, 6.20)	(2.50, 1.95, 5.82)
(0.13, 0.28, 0.27, 0.12)	(1.98, 1.67, 6.98)	(1.38, 1.48, 6.56)
(0.72, 0.86, 0.92, 0.34)	(5.35, 0.98, 2.57)	(2.01, 4.29, 3.58)
(0, 0, 0, 0)	(0.22, 0.00, 7.35)	(0.00, 0.00, 7.00)
(0.46, 0.12, 0.67, 0.45)	(3.79, 0.29, 5.88)	(3.11, 1.48, 5.20)
(0.35, 0.31, 0.54, 0.65)	(4.01, 1.80, 5.51)	(2.85, 1.76, 5.16)
(0.84, 0.35, 0.73, 0.23)	(3.79, 2.04, 5.47)	(1.57, 3.12, 5.18)
(0.23, 0.21, 0.32, 0.05)	(2.33, 0.78, 7.10)	(1.40, 1.31, 6.61)

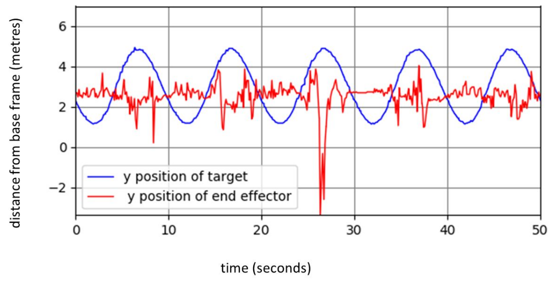
Accuracy is satisfactory for the Z position (maximum error is 1.01m, average error is 0.47m), indicating that the Forward Kinematics equation could be correct for the Z position of the end effector and suggesting that the major mistake causing an almost unpredictable offset in X and Y is a mathematical error in our FK model. We tried calculating a total of 3 FK variants, each with a different DH table and robot frame assignment, yet our best one is illustrated in our PDF.

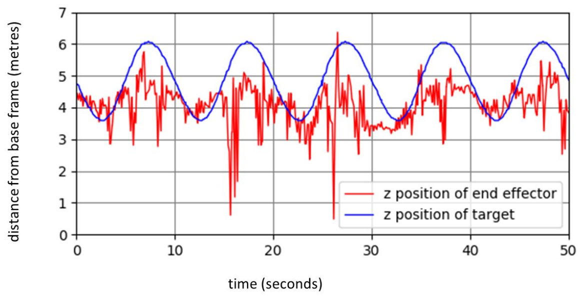
3.2 Closed-loop control

The controller uses a version of the closed-loop PD controller from Lab 3, adapted to work in 3D. After trying several different combinations of values for proportional and derivative gain, the values chosen to give the best results were a proportional gain of 5 and a derivative gain of 1. The Jacobian was calculated from our Forward Kinematics equation above, to be used in the controller.

Figures of x, y, z position of end-effector vs target







The **accuracy** of the controller is poor, with a major source of error being the incorrect calculation of the Forward Kinematics equation (and hence the Jacobian), further discussed in the section below. The x position has the greatest error - with the end effector position always negative while the target is on the positive side. The y and z position are closer to the target but display large spikes which indicate the robot is moving large distances suddenly - this can be a problem, in the real world these sudden movements are dangerous. We could reduce the occurence of this by introducing a threshold for the change in joint angles at each step, only allowing small changes at a time. This issue may be occurring due to the issue of the hidden blobs mentioned above, that we have not solved in our code.

General observations

The system is behaving in a very **unpredictable manner** for the most part, most probably because of erroneous calculations of the FK equations (as suggested by the table in 3.2). In our case, target prediction isn't a problem as our template matching solution performs very well (as observed in the figures on page 2 - the X positions are almost identical).

When the **target (orange sphere) is lost** in one of the cameras, we avoid instability by freezing the X/Y coordinate at the last seen position, and relying on the other camera to update our estimate of the Z position.

If the system **couldn't find the valid target every time**, an important factor which could cause this in our case is when the sphere is hidden and our algorithm switched to the box instead. To fix this, we set a threshold, and if the result of template matching fell below this, the sphere must be hidden, and we used the 3D target history instead. This means the estimate is slightly off, but much closer to the actual position than if it detected the box instead.

If given **more time**, we would seek support and guidance from a professional with regard to the correctness of our FK/Jacobian calculations. This should resolve our issues and allow the robot to perform in a predictable and satisfactory manner, as our codebase is well adapted to accommodate the fixes.