IVR COURSEWORK

Amy Rafferty s1817812 – Alasdair Forbes s1849401

Contributions:

- 2.1: Code – Alasdair, Report – Amy
- 2.2: Code - Both, Report – Amy
- 3.1: Calculating K Matrix – Both, Generating Values – Alasdair, Report – Amy
- 3.2: Calculating Jacobian – Alasdair, Generating Graphs – Alasdair – Both

Repository: https://github.com/aforbes23/IVR_assignment

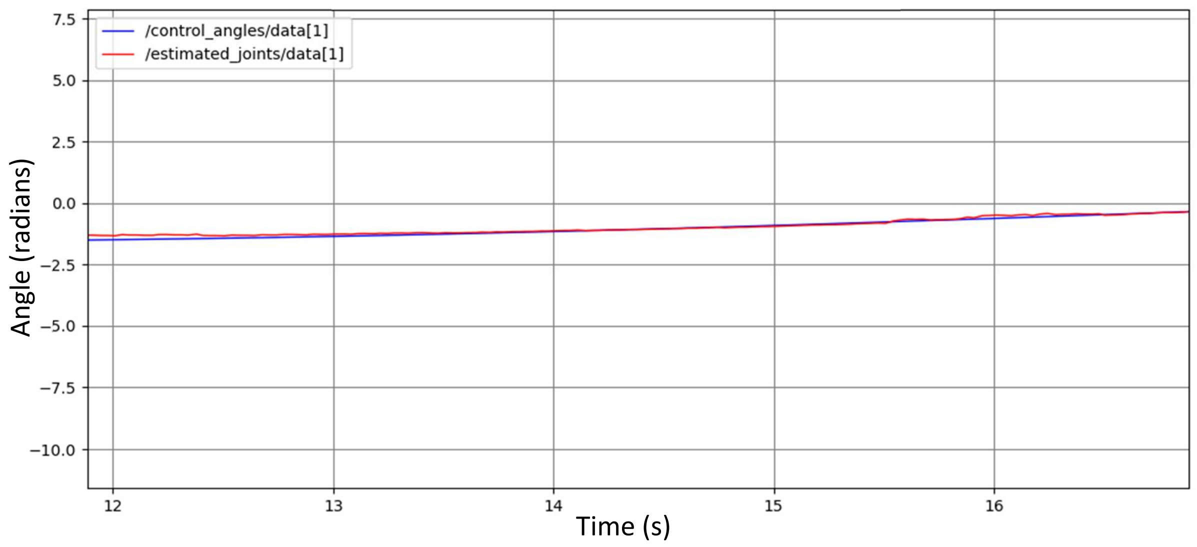
Question 2.1

Our algorithm first gets the coordinates of the centres of each joint and the end effector, by using blob detection for each joint colour (yellow, blue, green, red). During this step, the image is converted to HSV, as it tends to be a more reliable format than the default BGR for colour masks. Before any angles can be calculated, the coordinates of the blob centres must be converted from pixels to meters. The conversion factor between these metrics is found by dividing the length of a link in meters, as given in the coursework specification, by the distance between the centres of the blobs at each end of this link in the image. The coordinates of the centres are then converted to meters, with respect to the position of joint 1, which we take as the centre of the image (0,0).

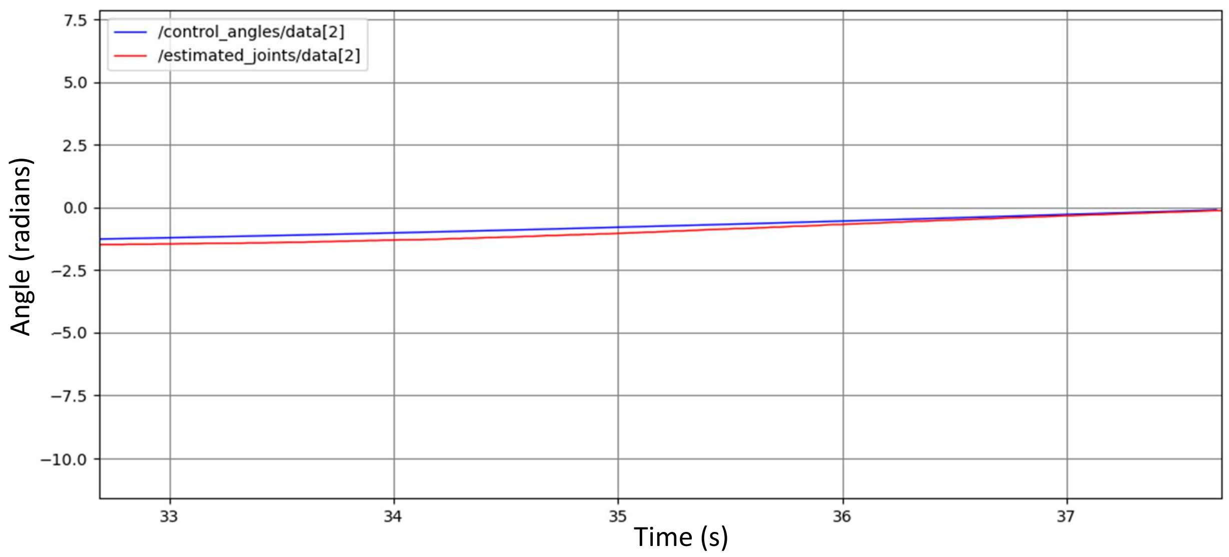
This is repeated for both cameras, which gives the result of a set of coordinates for each blob for each image. The coordinates for image 2 are published by image2.py and subscribed to by image1.py, which is where our algorithm primarily operates. Then, for each blob, these two sets of coordinates are combined to find the 3d position of the blob (x, y, z). Image 1 provides the x coordinate, image 2 provides the y coordinate, and the z coordinate should be the same for each image, so can be taken from either, providing that it is a legal number (not NaN). Then, trigonometry is used to find the estimated angles for the joints, each calculated on the plane in which the joint operates. For joint 2, this is the y-z plane, as it rotates around the x axis. Similarly, for joint 3, which operates on the x-z plane, and rotates around the y axis. Joint 4 is slightly different, as it does not operate on an axes plane. Instead, we had to calculate the rotation matrix for the plane, then transform the direction vector for the link to get the required angle.

These estimated angles are then published. The actual angles of the joints are calculated and published by image2.py, in the form of a sinusoidal trajectory specified in the coursework specification. Here, we have graphs showing the differences between these actual angles (shown in blue), and our estimated angles (shown in red), as found by our algorithm, each over a 5 second period.

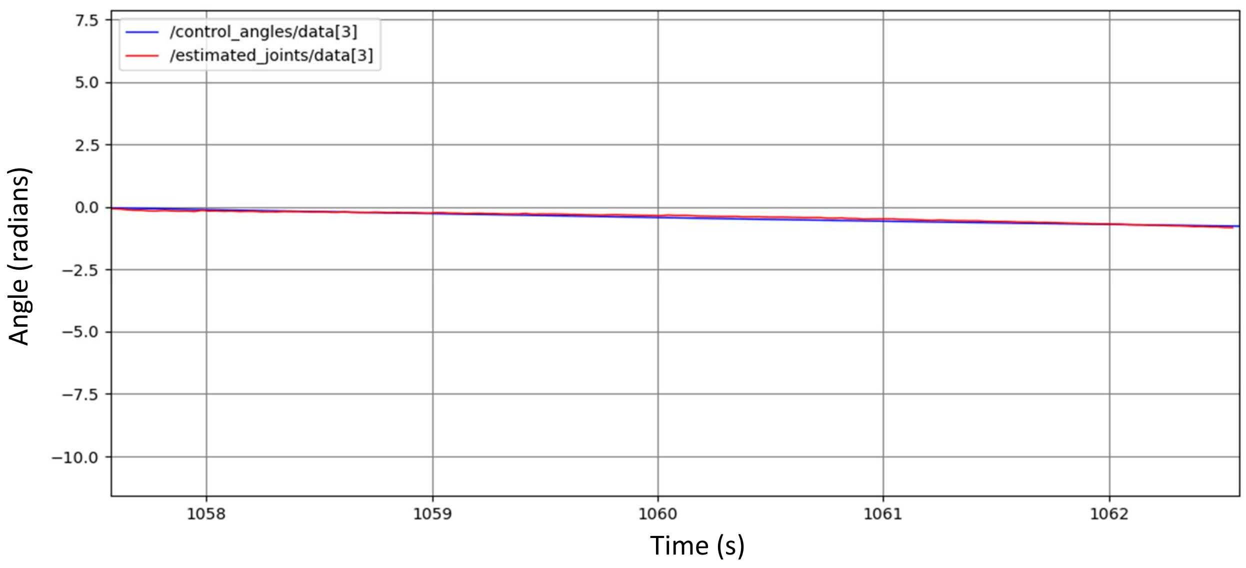
Actual and Estimated Trajectories for Joint 2



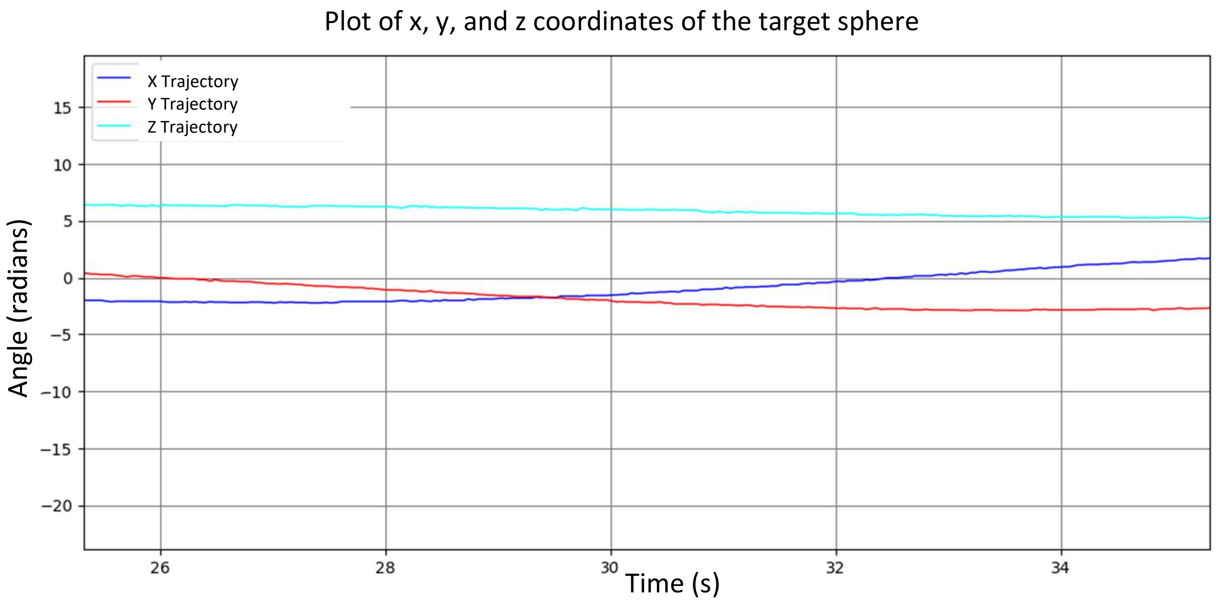
Actual and Estimated Trajectories for Joint 3



Actual and Estimated Trajectories for Joint 4



Question 2.2



When finding the target, the image is converted to HSV, and then the orange parts of the image are isolated. The original image is then converted to grayscale, and then masked to create a grayscale image where the orange areas are again isolated, so that OpenCV's HoughCircles method can detect the orange circles in the image. The circle found by this image will be the orange circle in the original image, which is the target. The (x, y) coordinates of the centre of this circle are then returned. This is repeated for the images received by both cameras in order to get the position of the target in the 3d space, by converting the positions from pixels to meters, and combining the x, y and z coordinates, as we did in part 1.

The sources of error in our measurements come when the target sphere is obscured by the orange rectangle, as it becomes difficult to differentiate the two objects of the same colour when they are overlapping. This is known as occlusion. Occlusion also occurs when the target sphere is obscured by the robot itself, as the algorithm cannot detect a circle that is not visible.

Question 3.1

Our Forwards Kinematics equation is as follows, where $s\theta_n$ represents $\sin\theta_n$, and $c\theta_n$ represents $\cos\theta_n$.

$$\begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} =$$

$$\begin{bmatrix} 3s\theta_3c\theta_4c\theta_1 + 3.5s\theta_3c\theta_1 - 3s\theta_4c\theta_2s\theta_1 - 3c\theta_3c\theta_4s\theta_1s\theta_2 - 3.5c\theta_3s\theta_1s\theta_2 \\ -3s\theta_3c\theta_4s\theta_1 - 3.5s\theta_3s\theta_1 - 3s\theta_4c\theta_1c\theta_2 - 3c\theta_3c\theta_4c\theta_1s\theta_2 - 3.5c\theta_3c\theta_1s\theta_2 \\ -3s\theta_4s\theta_2 + 3c\theta_3c\theta_4c\theta_2 + 3.5c\theta_2c\theta_3 + 2.5 \end{bmatrix}$$

We tested our result by comparing the calculated end effector positions to the actual positions, shown in the following table:

Comparison	Actual Position (x, y, z)	Calculated Position (x, y, z)
1	(1.87, -1.78, 8.91)	(1.73, -1.63, 8.52)
2	(-0.84, 2.07, 8.11)	(-1.11, 2.26, 8.44)
3	(0.58, 5.35, 6.16)	(1.57, 5.19, 4.89)

4	(4.54, -1.33, 7.43)	(5.10, -1.17, 6.30)
5	(4.71, 4.08, 4.75)	(5.01, 3.63, 4.22)
6	(0.12, -6.05, 0.36)	(-1.11, -2.01, 0.62)
7	(-6.02, 0.20, 5.64)	(-5.85, -0.03, 5.31)
8	(-1.20, 5.95, 5.24)	(-0.85, 2.96, 1.23)
9	(-2.15, 0.31, 8.46)	(-2.47, 0.19, 8.44)
10	(6.71, -6.62, 3.96)	(6.44, -0.14, 3.07)

As you can see, our forward kinematics equations are not as accurate as they could be. Some of the comparisons are successful, like comparisons 1, 2 and 9, where the values of x, y, and z are all similar. However, there are cases where our calculations seem to have encountered an issue, for example in comparison 6 where the y values are significantly different, and comparison 8, where the y and z values are significantly different. Our equation seems to be most accurate in the x axis, with no discrepancies higher than 1.5 (we appreciate that this is still a fairly significant discrepancy).

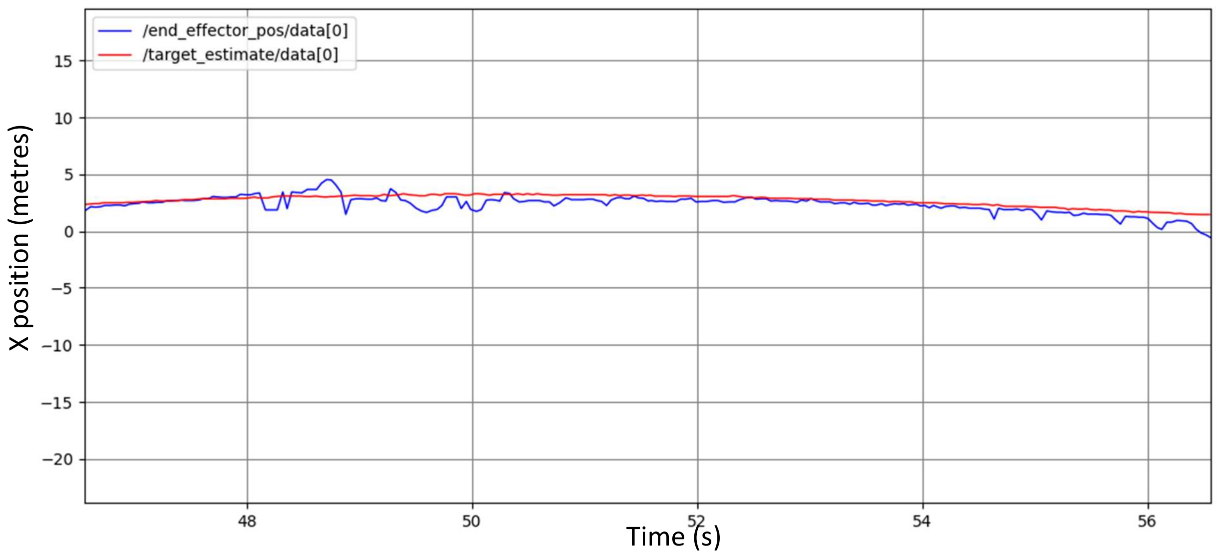
Question 3.2

The Jacobian of our robot is as follows, where:

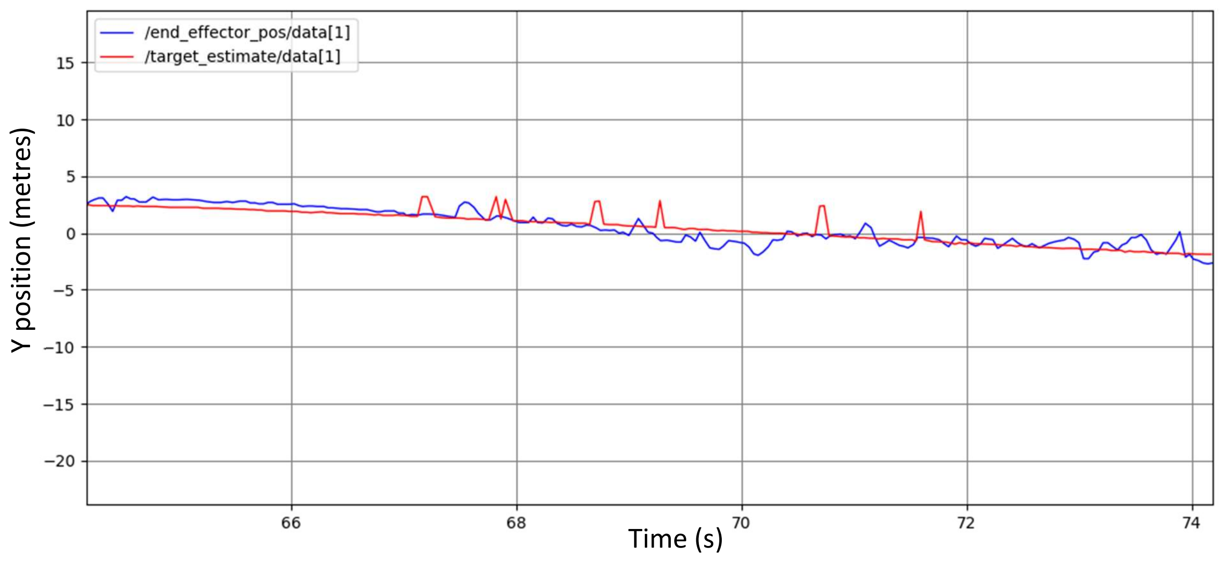
a = $sin\theta_1$, b = $cos\theta_1$, c = $sin\theta_2$, d = $cos\theta_2$, e = $sin\theta_3$, f = $cos\theta_3$, g = $sin\theta_4$, h = $cos\theta_4$

$$\begin{bmatrix} -3eha - 3.5ea - 3gdb - 3fhbc - 3.5fbc & 3gca - 3fhad - 3.5fad & 3fhb + 3.5fb + 3ehac + 3.5eac & -3egb - 3hba + 3fgac \\ -3ehb - 3.5eb + 3gad + 3fhac + 3.5fac & 3gbc - 3fhbd - 3.5fbd & -3fha - 3.5fa + 3ehba + 3.5ebc & 3ega - 3hbd + 3fgbc \\ 0 & -3gd - 3fhc - 3.5cf & -3ehd - 3.5de & -3hc - 3fgd \end{bmatrix}$$

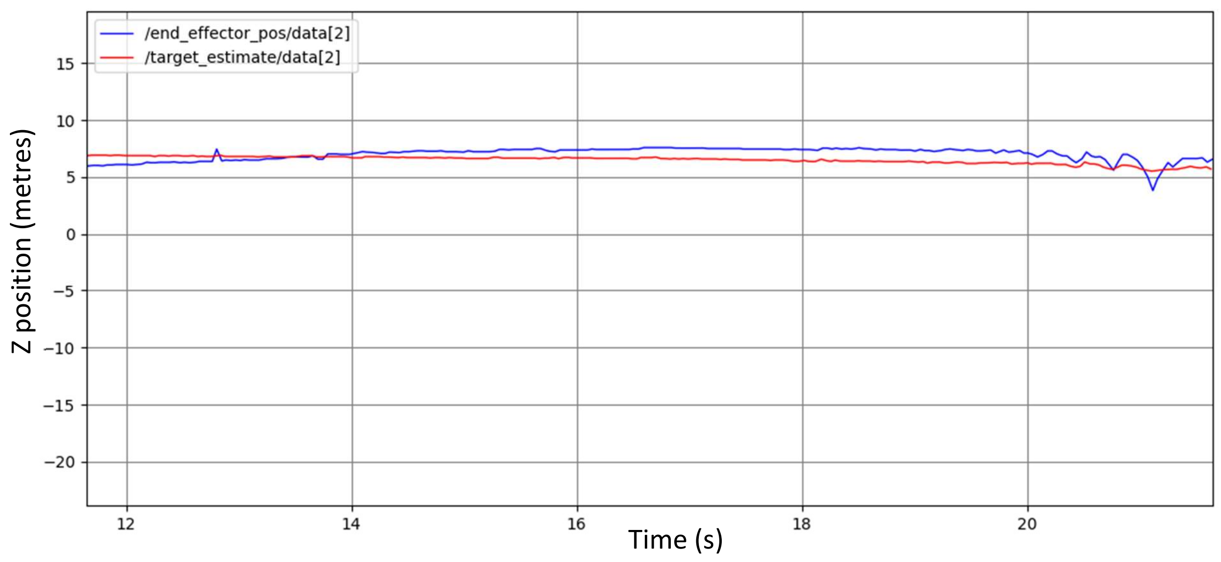
Plot of x of end effector and target in open-loop



Plot of y of end effector and target in open-loop



Plot of z of end effector and target in open-loop


/end_effector_pos/data[2]
/target_estimate/data[2]
Z position (metres)
Time (s)