IVR Coursework

Matt Timmons-Brown & Neil Weidinger

December 4, 2020

Neil and Matt worked on this coursework collaboratively - answering all questions and coding together. Thus there was an equal contribution from both members of the team.

Access the GitHub link to our code here: `https://github.com/the-raspberry-pi-guy/IVR-Assignment`

2 Robot Vision

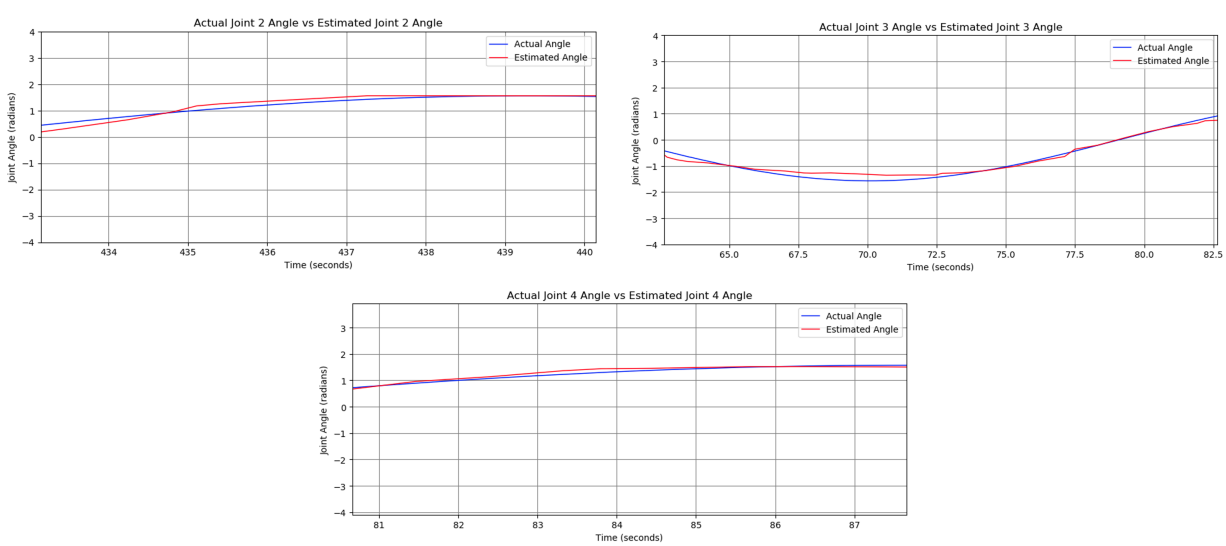
2.1 Joint State Estimation

The algorithm for joint state estimation is implemented over image1.py, image2.py and joint_target_estimation.py. Each of the image files subscribe to the orthogonal camera raw data streams that are pointed at the robot in Gazebo, and thus give a view of the x/z and y/z planes respectively.

When an image is received from the camera, the data from the image is passed to the callback - colour-based computer vision is then applied to detect each of the (different colour) joints of the robot. To achieve this, we used OpenCV colour masking, image dilation and then calculated the appropriate moment for the x/z or y/z coordinate pair, which are then published.

joint_target_estimation.py receives these coordinates and, as the cameras can be assumed to be orthogonal (according to Prof Khadem live class), the x/z and y/z coordinates for all joints can be combined to form an $[x, y, z]$ vector that represents the location of the joint in 3D space.

From this, the vectors between each link of the robot are determined (ie: the blue-green link is the blue vector minus the green vector). The arctan2 operator between the blue-green vector's y and z coordinates is then used to calculate the angle for joint 2. After this, a rotation operation with a rotation matrix around the x axis is applied to rotate the blue-green vector into the newly oriented x-axis frame. Joint 3 is then calculated with arctan2 between the rotated blue-green vector's x and z coordinates. Finally, joint 4 is calculated by applying arctan2 to the green-red joint vector's y and z coordinates, and then subtracting the joint 2 angle previously calculated - as these are in the same direction (x).



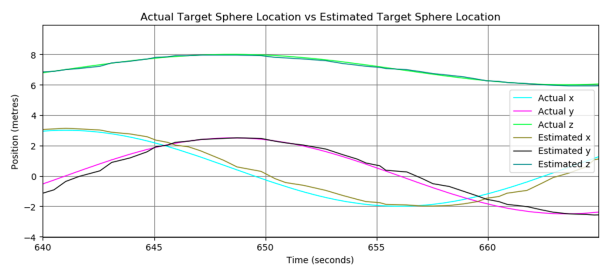
2.2 Target Detection

Similar to Q2.1, the algorithm for detection of the target sphere is distributed over the 3 files previously discussed. Crucially, there is further image processing in the image1.py and image2.py files in order to identify the sphere specifically (as it is the same colour as the cube).

As before, when an image is received from a camera, the callback triggers a detection algorithm. This disregards the bottom half of the image (as the sphere and cube never appear here) and then creates a mask with the orange shapes cut out. The contours of these shapes are then found using OpenCV, and the number of sides of the shape are identified by approximating a polygon to each shape with the OpenCV implementation of the Douglas-Peucker algorithm. The shape with the most sides is the sphere, as the cube only ever has approximately 4 sides according to our CV testing.

The moment of this shape is then calculated as before, and this happens in each camera to provide x/z and y/z coordinates as described in Q2.1. These are combined as before to form a $[x, y, z]$ estimate of the pixel location of the target sphere. Finally, a pixels-to-metres operation is performed to convert the pixel location to a metre-location with respect to the robot base frame. A scaling factor is calculated by counting the number of pixels between the base and first joint of the robot, and then dividing this by the known length (2.5m) of this link. We later hard coded this scaling factor for quick conversion of values.

There are a few sources of error in our measurements. Most notably, occasionally the target sphere is occluded by the robot from the perspective of one of the camera's viewpoints (ie when it orbits behind the robot). This



3 Robot Control

3.1 Forward Kinematics

Insert table of DH params?

$$\begin{bmatrix} 3s(\theta_1)s(\theta_2)c(\theta_3)c(\theta_4) + 3.5s(\theta_1)s(\theta_2)c(\theta_3) + 3s(\theta_1)s(\theta_4)c(\theta_2) + 3s(\theta_3)c(\theta_1)c(\theta_4) + 3.5s(\theta_3)c(\theta_1) \\ 3s(\theta_1)s(\theta_3)c(\theta_4) + 3.5s(\theta_1)s(\theta_3) - 3s(\theta_2)c(\theta_1)c(\theta_3)c(\theta_4) - 3.5s(\theta_2)c(\theta_1)c(\theta_3) - 3s(\theta_4)c(\theta_1)c(\theta_2) \\ -3s(\theta_2)s(\theta_4) + 3c(\theta_2)c(\theta_3)c(\theta_4) + 3.5c(\theta_2)c(\theta_3) + 2.5 \end{bmatrix}$$

Where $c(\theta_i) = \cos(\theta_i)$, $s(\theta_i) = \sin(\theta_i)$

Joint Angle *Joint 1,2,3,4 (rad)*	Estimated via FK *x,y,z (m)*	Estimated via Images *x,y,z (m)*
1,0.5,0.1,-1	0.47,0.31,8.18	0.33,0.33,8.76
-1,-1,-1,1	-1.52,4.15,6.12	-1.36,3.68,6.59
0.25,0.25,0.25,0.25	2.09, -1.79, 8.33	2.24,-2.06,9.16
1,1,0.5,0.5	6.05,-0.39,4.20	6.11,-0.59,4.64
-1,-0.5,-0.1,1	4.20,2.09,5.76	3.68,2.72,6.22
1,1,1,-1	3.14,3.10,6.12	2.54,3.72,6.77
-0.25,-0.25,-0.25,-0.25	-0.98,2.58,8.33	-0.92,2.43,8.65
-1,-1,-0.5,-0.5	2.88,5.34,4.20	2.13,6.33,4.97
π, $\pi/2$, $\pi/4$, -0.1	-4.58,4.59,2.80	-3.72,3.64,3.86
$-\pi$, $-\pi/2$, $-\pi/4$, 0.1	4.59,-4.58,2.80	6.07,-6.15,2.43

COMMENT ON ACCURACY

3.2 Closed-Loop Control

$$A =$$

$$\begin{bmatrix} -3s(\theta_1)s(\theta_3)c(\theta_4) - 3.5s(\theta_1)s(\theta_3) + 3s(\theta_2)c(\theta_1)c(\theta_3)c(\theta_4) + 3.5s(\theta_2)c(\theta_1)c(\theta_3) + 3s(\theta_4)c(\theta_1)c(\theta_2) \\ 3s(\theta_1)s(\theta_2)c(\theta_3)c(\theta_4) + 3.5s(\theta_1)s(\theta_2)c(\theta_3) + 3s(\theta_1)s(\theta_4)c(\theta_2) + 3s(\theta_3)c(\theta_1)c(\theta_4) + 3.5s(\theta_3)c(\theta_1) \\ 0 \end{bmatrix}$$

$$B =$$

$$\begin{bmatrix} -3s(\theta_1)s(\theta_2)s(\theta_4) + 3s(\theta_1)c(\theta_2)c(\theta_3)c(\theta_4) + 3.5s(\theta_1)c(\theta_2)c(\theta_3) \\ 3s(\theta_2)s(\theta_4)c(\theta_1) - 3c(\theta_1)c(\theta_2)c(\theta_3)c(\theta_4) - 3.5c(\theta_1)c(\theta_2)c(\theta_3) \\ -3s(\theta_2)c(\theta_3)c(\theta_4) - 3.5s(\theta_2)c(\theta_3) - 3s(\theta_4)c(\theta_2) \end{bmatrix}$$

$$C =$$

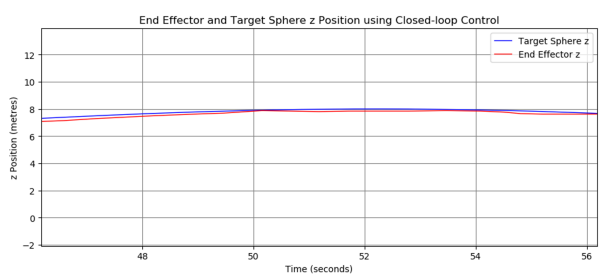
$$\begin{bmatrix} -3s(\theta_1)s(\theta_2)s(\theta_3)c(\theta_4) - 3.5s(\theta_1)s(\theta_2)s(\theta_3) + 3c(\theta_1)c(\theta_3)c(\theta_4) + 3.5c(\theta_1)c(\theta_3) \\ 3s(\theta_1)c(\theta_3)c(\theta_4) + 3.5s(\theta_1)c(\theta_3) + 3s(\theta_2)s(\theta_3)c(\theta_1)c(\theta_4) + 3.5s(\theta_2)s(\theta_3)c(\theta_1) \\ -3s(\theta_3)c(\theta_2)c(\theta_4) - 3.5s(\theta_3)c(\theta_2) \end{bmatrix}$$

$$D =$$

$$\begin{bmatrix} -3s(\theta_1)s(\theta_2)s(\theta_4)c(\theta_3) + 3s(\theta_1)c(\theta_2)c(\theta_4) - 3s(\theta_3)s(\theta_4)c(\theta_1) \\ -3s(\theta_1)s(\theta_3)s(\theta_4) + 3s(\theta_2)s(\theta_4)c(\theta_1)c(\theta_3) - 3c(\theta_1)c(\theta_2)c(\theta_4) \\ -3s(\theta_2)c(\theta_4) - 3s(\theta_4)c(\theta_2)c(\theta_3) \end{bmatrix}$$

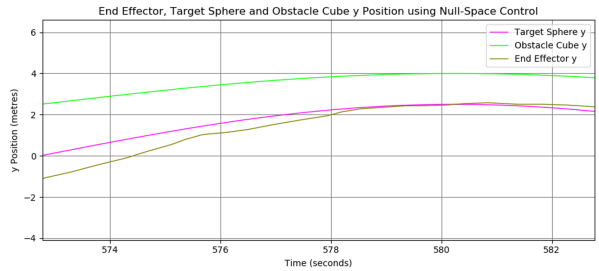
Where $c(\theta_i) = \cos(\theta_i)$, $s(\theta_i) = \sin(\theta_i)$ and A, B, C, D are column vectors that form the Jacobian J when arranged like (formatted to save space):

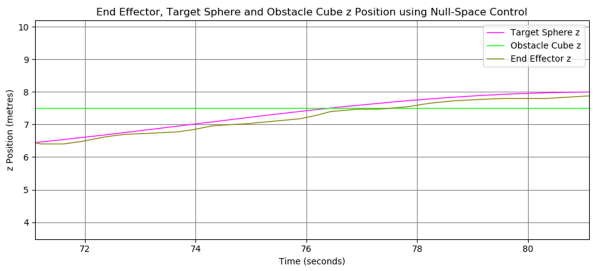
$$J = \begin{bmatrix} A & B & C & D \end{bmatrix}$$

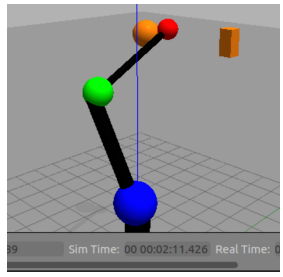


4 Final Task

4.2 Null-space Control







The previous closed-loop controller attempted to move the robot end effector to track the spherical target as closely as possible. This null-space controller is quite similar in that it also attempts to track the spherical target, but it also has a secondary goal of attempting to avoid hitting the cube target as far as is possible without compromising the original goal of tracking the spherical target.

Our robot is a redundant system: it has more DOF than is needed to complete the task of tracking the sphere. This is because both joint 2 and joint 4 rotate around the x axis, meaning we can achieve the same end effector position using two different sets of angles for these joints (at least for the task to position the end effector close to the sphere, not quite redundant if we would like to move the end effector to more extreme positions). In other words, it is possible to accomplish our primary goal of tracking the spherical target in multiple ways.

Using this knowledge, we can send also control signals to achieve our secondary goal of avoiding the cube target without affecting the performance of our primary goal. The way this works is by