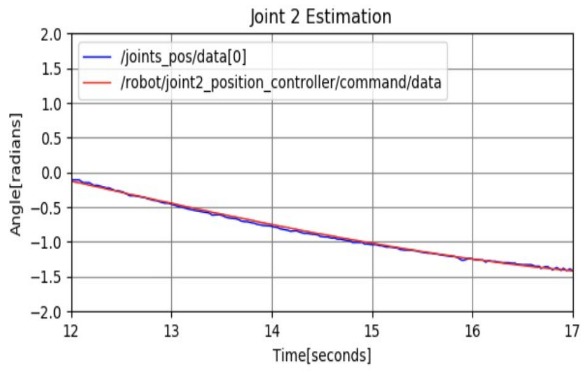
IVR Assignment
s1813106 and s1741043

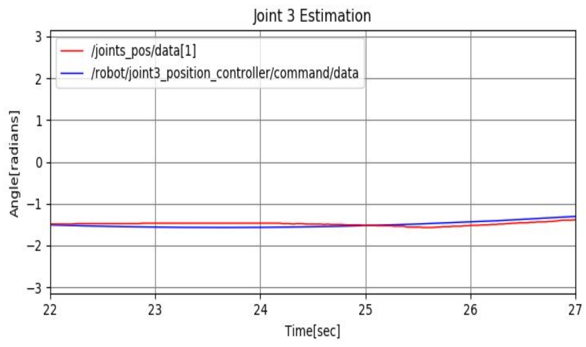
Who has worked on what part of the project:
Part2 and Part4.1 done by s1813106. Part 3 done by s1741043.
Link to your GitHub account for downloading your final ROS library:
https://github.com/orgesskura/IVR_assignment

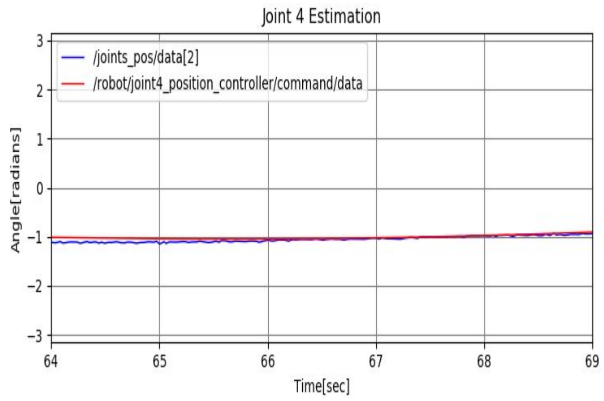
Q2.1) Joint State Estimation (15 marks)

- Describe your algorithm (3 marks):

In my algorithm I first detect red,blue, yellow and green via moments in a similar fashion to lab3. I have also added a piece of code that states that if one of the spheres is obscured in one of the cameras the algorithm returns its last observed center. This helps as some spheres may be obscured in some instances in one of the cameras. I hardcode pixel to meter conversions as that might help to make the algorithm faster. Also, I have hardcoded the position of yellow and blue since they do not change position. Next, I have written functions to derive x,y,z positions of spheres in the real world. To find angle of each joint I do:
Joint 2 : I just take arctan of positions of z and y taken from camera 1 of the link from green to blue. I do that as an assumption that rotations in y will not change the angles in camera 1.
Joint3: I used the same method as joint2 but this time I took x and z from camera 2 and applied the arctan. I assumed that rotations around x in x-z plane do not change angles in that plane.
Joint4 : First I wanted to use the same method as Joint 2 but got wrong results and joint 4 was dependent on joint2. So then, I just find the angle between the vector from blue to green and the vector from green to red. This works as blue to green has been rotated in x and y while green 2 red has been rotated in x,y,x. I worked out some edge cases and found out which negations work best for the images.



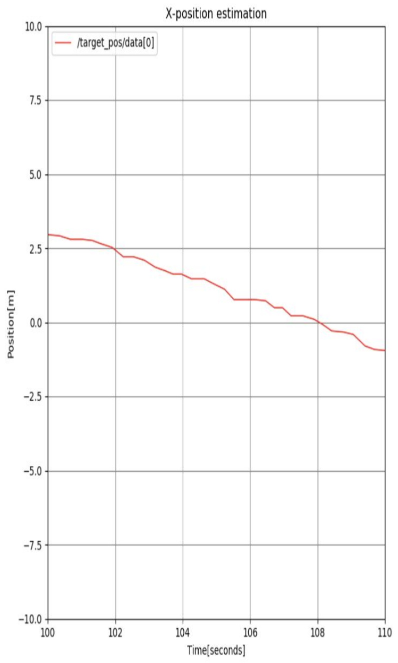


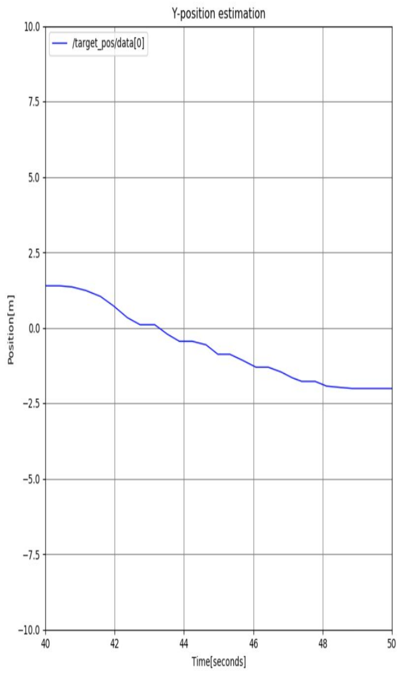


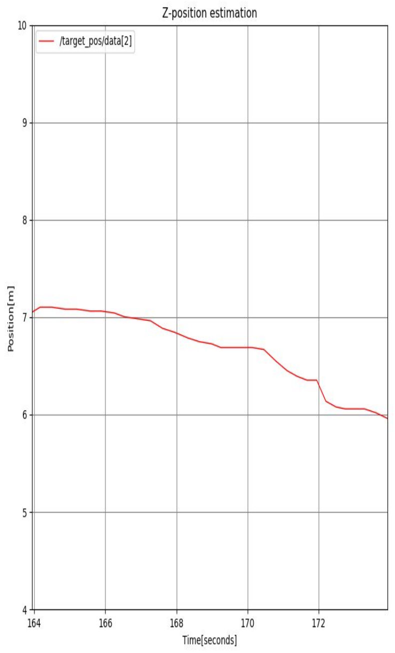
Q2.2) Target Detection (5 marks)

- Describe your algorithm and comment on the sources of error in your measurements (2 marks):

First I applied thresholding to get only the orange objects. Then I cropped the orange sphere to use as a template. Next I applied the template matching to get the sphere and not the box. Then, using the same method as q2.1, I get the x,y,z coordinates of the orange sphere with respect to the robot base frame. We can see that they follow a nice trajectory. The only problem is the z coordinates which are shifted by about 1. I expect this is happening because the robot is mostly in the upper part of the image causing the conversion from pixel to meters to be less accurate. Another error may be that parts of sphere are not visible in one of the cameras. Also, I deal with the case of the sphere not being visible at all in the same way as in Q2.1.







Q3.1) Forward Kinematics

$A^0{}_1 =$

$$\begin{bmatrix} -\sin(\theta_1) & 0 & \cos(\theta_1) & 0 \\ \cos(\theta_1) & 0 & \sin(\theta_1) & 0 \\ 0 & 1 & 0 & 2.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$A^1{}_2 =$

$$\begin{bmatrix} -\sin(\theta_2) & 0 & \cos(\theta_2) & 0 \\ \cos(\theta_2) & 0 & \sin(\theta_2) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$A^2{}_3 =$

$$\begin{bmatrix} \cos(\theta_3) & 0 & -\sin(\theta_3) & 3.5\cos(\theta_3) \\ \sin(\theta_3) & 0 & \cos(\theta_3) & 3.5\sin(\theta_3) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$A^3{}_4 =$

$$\begin{bmatrix} \cos(\theta_4) & -\sin(\theta_4) & 0 & 3\cos(\theta_4) \\ \sin(\theta_4) & \cos(\theta_4) & 0 & 3\sin(\theta_4) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

DH Table:

	α	a	d	θ
1	π/2	0	2.5	Θ1+π/2
2	π/2	0	0	θ2+π/2
3	-π/2	3.5	0	θ3
4	0	3	0	θ4

$A^0{}_4 = A^0{}_1 A^1{}_2 A^2{}_3 A^3{}_4$

[x y z] =

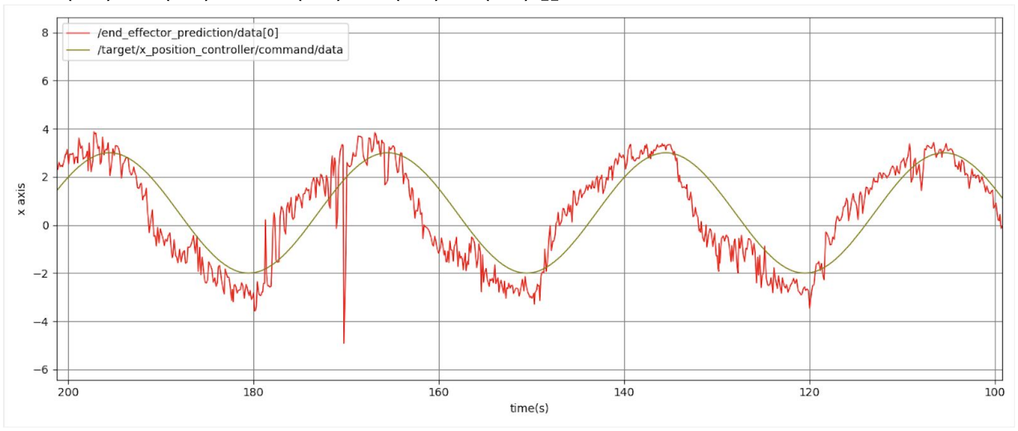
[3(sin(θ1)sin(θ2)cos(θ3) + sin(θ3)cos(θ1))cos(θ4) + 3.5sin(θ1)sin(θ2)cos(θ3)
+3sin(θ1)sin(θ4)cos(θ2) + 3.5sin(θ3)cos(θ1) ,
3(sin(θ1)sin(θ3) - sin(θ2)cos(θ1)cos(θ3))cos(θ4) + 3.5sin(θ1)sin(θ3)
-3.5sin(θ2)cos(θ1)cos(θ3) -3sin(θ4)cos(θ1)cos(θ2) ,
-3sin(θ2)sin(θ4) + 3cos(θ2)cos(θ3)cos(θ4) + 3.5(cos(θ2)cos(θ3) + 2.5]
End Effector Position Table

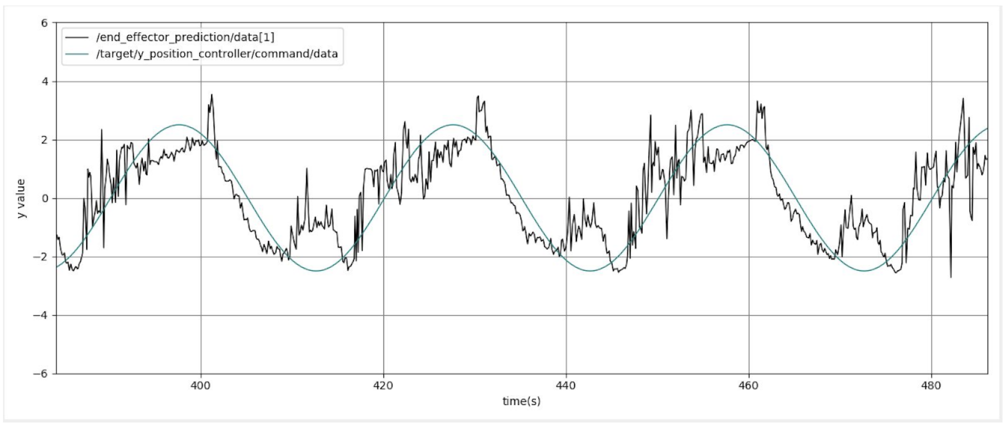
Estimated via vision (x,y,z) in meters	Estimated via FK (x,y,z) in meters	10 Different Points In the format (theta1,theta2,theta3,theta4) In radians
(4.7,-1.06,6.15)	(4.3,-0.96,6.11)	(0,1,1,-1)
(5.7,-5.22,0.23)	(4.33,-3.7,1.87)	(0,1,1,1)
(-0.03,-3.67,5.44)	(0,-3.5,5.5)	(0,1.57,0,-1.57)
(5.28,-2.89,6.67)	(4.42,-1.96,6.53)	(0.5,0.5,0.5,0.5)
(3.32,0.395,8.14)	(3.21,0.25,7.9)	(0.5,0.5,0.5,-0.5)
(0.59,5.37,6.259)	(0.73,4.78,6.53)	(0.5,-0.5,0.5,-0.5)
(-4.3,1.7,6.22)	(-4.42,1.96,6.53)	(0.5,-0.5,-0.5,-0.5)
(-0.62,4.98,6.26)	(-0.74,4.78,6.53)	(-0.5,-0.5,-0.5,-0.5)
(-5.32,-0.32,6.27)	(-4.82,-0.39,6.53)	(1,-0.5,-0.5,-0.5)
(-0.7,-3.28,8.16)	(-0.48,-3.19,7.91)	(1,0.5,-0.5,-0.5)

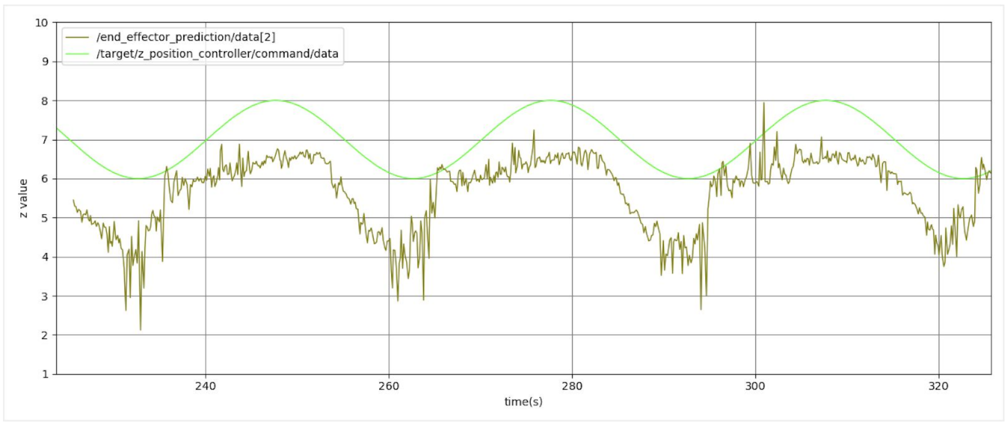
- Comment on the accuracy:
  The accuracy is generally good and within 0.5 error radians. In some cases like the second example the accuracy has dropped and is within an error of 1.5 radians.This might have happened because parts of the red object were not visible and the center of mass was calculated for the remaining part, meaning that center is shifted.
  One other error might come from the perspective of the camera, causing the conversion of pixels to meters to not be exact. I tried to improve the visibility error by returning the previous position in case the sphere is obscured by another sphere.

Q3.2) Closed loop Control
Jacobian (velocity kinematics calculation) :
[[3(cos(θ1)sin(θ2)cos(θ3) - sin(θ3)sin(θ1))cosθ4 + 3.5cos(θ1)sin(θ2)cos(θ3)
+3cos(θ1)sin(θ4)cos(θ2) - 3.5sin(θ3)sin(θ1),
3(sin(θ1)cos(θ2)cos(θ3))cos(θ4) + 3.5sin(θ1)cos(θ2)cos(θ3) -
3sin(θ1)sin(θ4)sin(θ2),
3(-sin(θ1)sin(θ2)sin(θ3) + cos(θ3)cos(θ1))cos(θ4) - 3.5sin(θ1)sin(θ2)sin(θ3) +
3.5 cos(θ3)cos(θ1),
-3(sin(θ1)sin(θ2)cos(θ3) + sin(θ3)cos(θ1))sin(θ4) + 3sin(θ1)cos(θ4)cos(θ2) **]**,
[3(cos(θ1)sin(θ3) + sin(θ2)sin(θ1)cos(θ3))cos(θ4) + 3.5cos(θ1)sin(θ3)
+ 3.5sin(θ2)sin(θ1)cos(θ3) + 3sin(θ4)sin(θ1)cos(θ2),
3(-cos(θ2)cos(θ1)cos(θ3))cos(θ4) - 3.5cos(θ2)cos(θ1)cos(θ3) + 3sin(θ4)cos(θ1)sin(θ2),
3(sin(θ1)cos(θ3) + cos(θ2)cos(θ1)cos(θ3))cos(θ4) + 3.5sin(θ1)cos(θ3) +
3.5sin(θ2)cos(θ1)sin(θ3),
-3(sin(θ1)sin(θ3)-sin(θ2)cos(θ1)cos(θ3))sin(θ4) - 3cos(θ4)cos(θ1)cos(θ2) **]**,

[0,
$-3\cos(\theta 2)\sin(\theta 4) - 3\sin(\theta 2)\cos(\theta 3)\cos(\theta 4) - 3.5\sin(\theta 2)\cos(\theta 3)$,
$-3\cos(\theta 2)\sin(\theta 3)\cos(\theta 4) - 3.5\cos(\theta 2)\sin(\theta 3)$,
$-3\sin(\theta 2)\cos(\theta 4) - 3\cos(\theta 2)\cos(\theta 3)\sin(\theta 4)$ **]]**







As we can see, the coordinates are quite close to the coordinates of the sphere. All measurements are pretty close except that the z coordinates are shifted by 1 but still follow the same trajectory. Since I have used the target coordinates from vision, this behaviour is expected as noted in Q2.2.

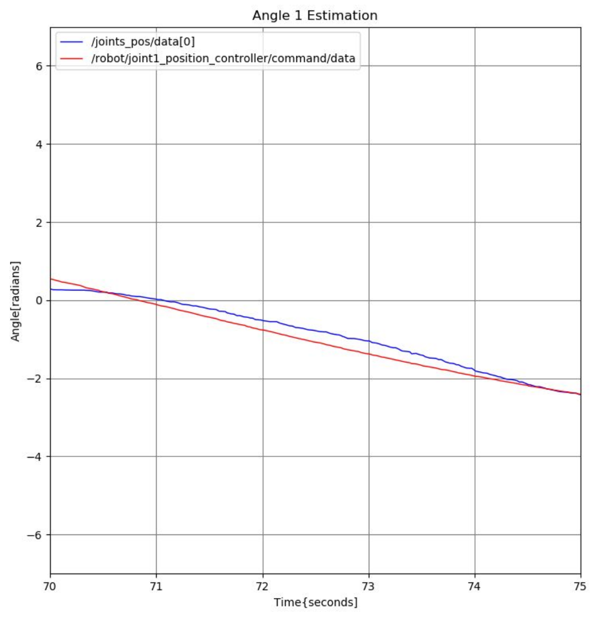
Q4.1)

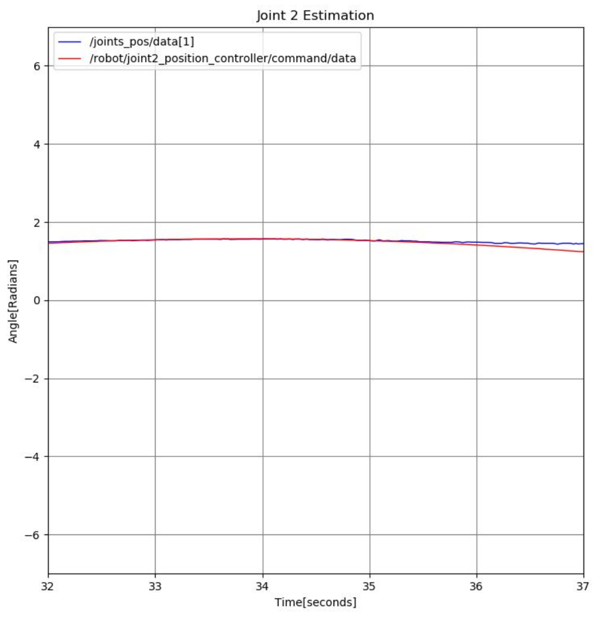
- describe your algorithm (2 marks) :

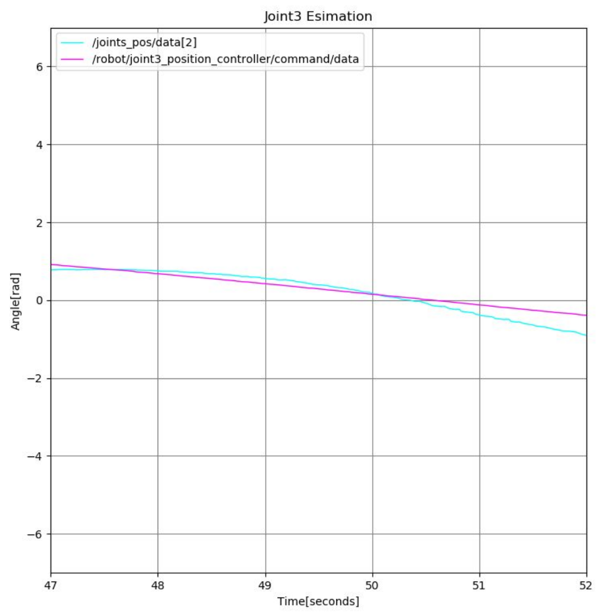
I used the same methods of detecting spheres as in Q2.1. For Joints I used:
Joint1: I just took the arctan of y and z coordinates from blue to green as rotations around z only change the x and y frame while keeping z frame the same.

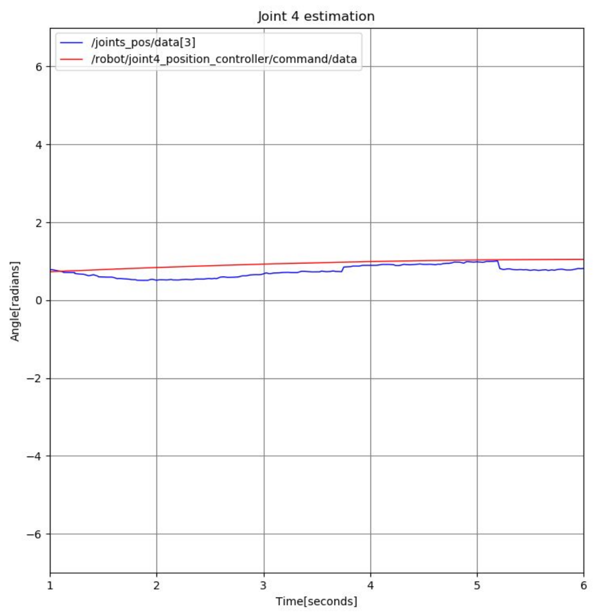
Joint2: Then to find the rotations around x I just use the arctan of z and y coordinates as those are the ones that change when rotated around x.
Joint3: First I rotate in the reverse direction of joint 2 to remove rotations around x. Then I use the z and y coordinates of the link from blue to green to estimate the joint3 angle.

Joint4 : To estimate joint4 I project link4 on link3 to remove effects of previous rotations. Then I try some edge cases to see which works and which not and then depending on that I assign the joint values.


Angle 1 Estimation
/joints_pos/data[0]
/robot/joint1_position_controller/command/data
Angle[radians]
Time{seconds]


Joint 2 Estimation
/joints_pos/data[1]
/robot/joint2_position_controller/command/data
Angle[Radians]
Time[seconds]


Joint3 Esimation
/joints_pos/data[2]
/robot/joint3_position_controller/command/data
Angle[rad]
Time[seconds]


Joint 4 estimation
/joints_pos/data[3]
/robot/joint4_position_controller/command/data
Angle[radians]
Time[seconds]