**Development**

2.1 Code: Si Han, Report: Both 2.2 Code: Si Han, Report: Saivydas

3.1 Code: Both, Report: Saivydas 3.2 Code: Both, Report: Saivydas

*GitHub:* [*https://github.com/SaiVillani/IVR*](https://github.com/SaiVillani/IVR)

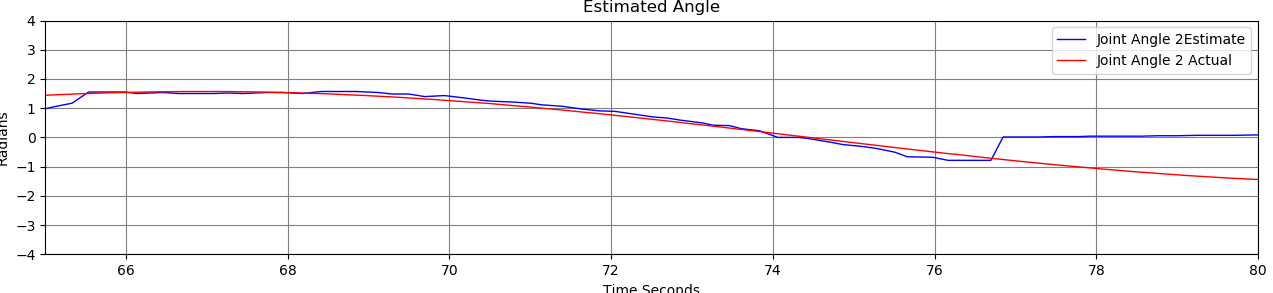
**Robot Vision**

2.1: Joint state estimation

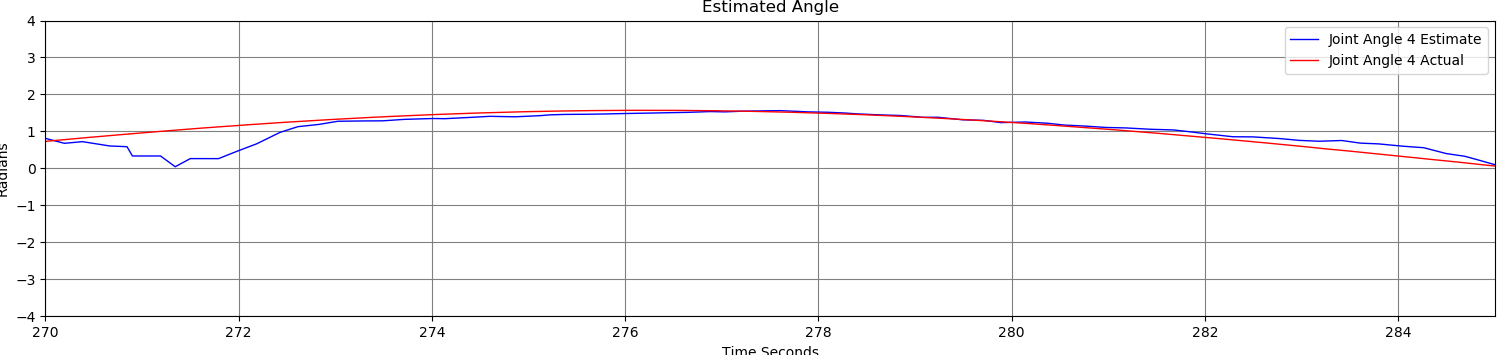
The algorithm first uses detects colour function via blob detection, to detect the relevant colour and zone in on the centre of the sphere, using BGR format for red, green, blue, and yellow. There are two callback functions, these allow us to tune in to the camera streams, as well as re-direct our cameras. For example, in callback2 when camera 2 fails to detect the red sphere the estimated created from the image of camera 1 are used instead. Beyond this, we use a standard pixel to meter conversion function that is used to detect the joint angles. 2 implementations for the detection of joint angles are included, a basic version which relies on trigonometry, and many if clauses for corner cases, which as you will see in the graphs result in violent pulls and flat lines, as well as a chamfer matching implementation that is built on a detecting the rotation of the links by matching the outline of the links to improve our estimations.

Given that Joint angle 4 was so difficult, it wasn’t enough to simply switch between camera 2 and camera 1 when the red sphere is not visible. This is because, even when camera 1 can see all 3 relevant components compared to camera 2 it still fails to detect motion in the x axis because the picture is completely adjacent to the robot. So, for camera 1 it is very difficult to tell if the red sphere and thus joint angle 4 is moving away or towards it. So, we attempted to implement linear extrapolation by estimating the position of the red sphere via our knowledge of the position of the blue and yellow spheres, which are much easier to spot, thus trigonometry can be used to estimate the position of the red sphere. This allows us to better deal with this mirroring effect of misinterpreting the sign of the angle, by transforming a given picture to our extrapolated picture via the orthogonal vector which allows us to map the coordinates from real to extrapolated.

The estimations of joint angle 2 are not optimal. The Chamfer matching estimations seem to be the most consistent, particularly on the slopes, but not near the troth of the curve, this is likely to be due to the obstruction of yellow and blue spheres.



Chart, line chart

Description automatically generatedThe estimation of joint angle 3 are significantly better for both the simpler method and Chamfer matching model. However, both of our estimation models will flatten out near the peak or troth of the actual angle, this is due a hard coded assumption, that when [Details] we know it must be a right angle. The estimation recovers from the hard code quickly. The estimation of joint angle 4 is the worst by far. The simple estimation method seems to completely oppose the sign of the curve, we encountered a similar problem in vision2 as well. The chamfer method seems to be better at picking up the correct direction, and occasionally follows closely to the true angle, however, it often wildly deviates from the actual pattern of motion. [details]. We did perform investigations, which included snap shots of the exact times when either camera would not be able to either see a sphere because it was blocked by another, or because it failed to capture the relevant axis of motion. These investigations are in the command text files.

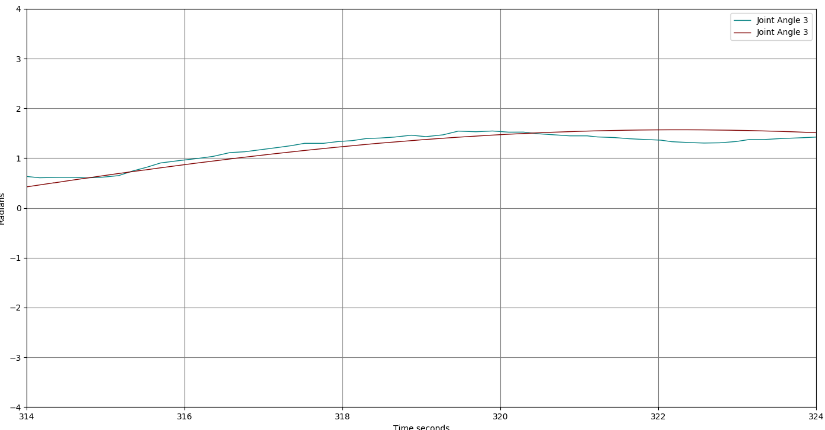
2.2: Joint state estimation

We adopted a similar algorithm which involves blob detection and trigonometric properties. However, it is a challenging task to estimate joint angle 1 and joint angle 3 due to the following reasons: 1) We could rotate joint angle 1 by an angle of pi and flip the signs of joint angle 3 to arrive the same set of positions 2) At any given joint angle 3, we could rotate joint angle 1 from pi to -pi (or vice versa) to reach the same set of positions. We started joint angle estimations through joint angle 1. Any angle rotation about the z-axis could be detected by finding the (x,y) coordinates of the blue sphere and comparing it with the angle it makes with the y-axis because the blue circle could only move along the y-axis when joint 3 rotates about the x-axis and joint 1 does not rotate. With this observation, we used the (x,y) coordinates detected from camera 1 (i.e. x coordinate of image = y coordinate in 3d space, y coordinate of image = z coordinate in 3d space) and camera 2 (i.e. x coordinates of image = x coordinate in 3d space, y coordinate of image = z coordinate of 3d space) for both yellow and blue spheres. This estimated acute angle, could then be adjusted to whichever quadrant the x-y coordinate holds the blue sphere. The bellow adjustments were made for preliminary estimates for joint angle 1 which we further adjusted after considering the joint angle 3 estimates.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Quadrants | Top Left | Top Right | Bottom Left | Bottom Right |
| Adjustment | 𝜋-Estimate Joint Angle1 | Estimate Joint Angle1 | - 𝜋 +Estimate Joint Angle1 | - Estimate Joint Angle1 |

Next, we estimated joint angle 3. We began by deciding which camera image to use by using this criterion: if camera 1 recorded a distance between the blue and yellow spheres to be within +/-0.5m of 3.2m, we would use camera 1’s image, else we check if camera 2 is within +/- 0.5m of 3.2. where 3.2m link length connecting blue circle and the yellow. This is a good heuristic to pick the leas-worst camera when neither have a clear view. After getting the initial estimates of joint angles 1 and 3, we consider them together. We noticed that if the blue or yellow spheres obstruct each other when captured by camera 1, the signs of polarity flip for joint angles 1 and 3. Detecting that the blue circle is in front or behind of another object is not easy, so we use the image of camera 2 to see if it is on the left or right side.

Chart, line chart

Description automatically generatedChart, line chart, scatter chart

Description automatically generatedWe rely on joint angle 1 and assume it is correct, then we adjust the sign of joint angle 3 according to where the blue circle is detected by camera 2. We use smoothing by considering the past trend using weighted average similar to exponentially weighted moving average method. We also kept track of the positive and negative increments in the estimates and their respective counts, we also used this to compute a forecast. We then checked to see if the weighted average is far from current joint angle 1 estimates. If they are deemed relatively close to the weighted average, we would check to see if it the trend is changing before reaching pi -pi and adjust them so that the estimate is smoother. If the estimates are far from the weighted average, we would try out the values +- pi away from the joint angle 1, picking the adjusted value closer to the weighted average and fine-tuning the results so that they are consistent with the trend. We also tried to smooth the joint angle 3 estimates by averaging the forecasted value and the estimates derived from image detection. Finally, we detected joint angle 4 by using the same approach in task 2.1. We used the linear "extrapolation" of where the red sphere should be at zero angle using the positions of the yellow and blue circle using camera 2. We computed the orthogonal projection of this vector on to the vector of the current red and blue spheres and applied arctan to get the joint angle estimate. To determine the sign of this angle, we transformed the existing picture coordinates. If the red sphere falls in the negative side of the new x-axis, we will flip the sign of the joint angle estimate.

**Robot Control**

*3.1: Forward Kinematics*

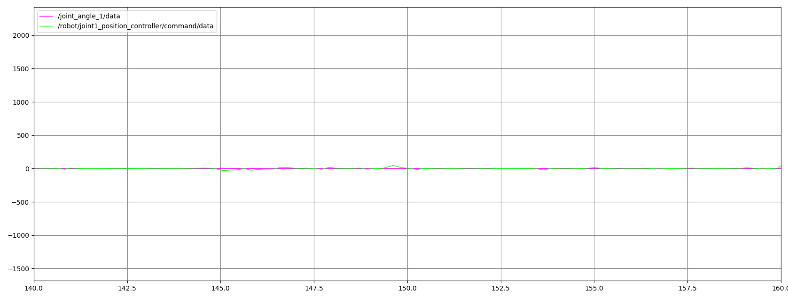
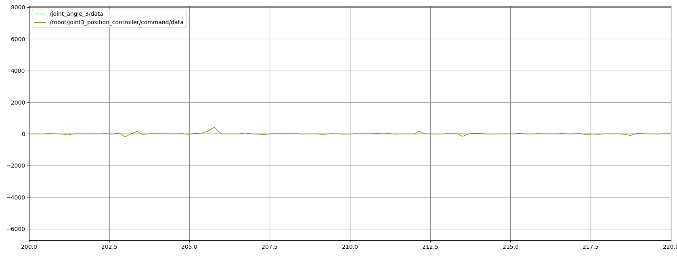
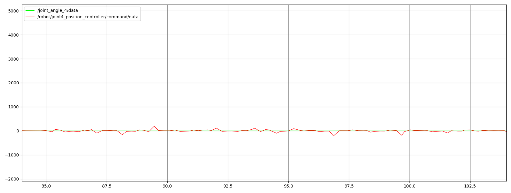
The algorithm uses the simple stuff that we learned from forward kinematics. The forward kinematics calculation was done with the following notation, as well as these Denavit-Hartenberg parameters, where d is the distance between the previous x-axis and current x-axis, θ is the angle around the z-axis between the previous x-axis and the current x-axis, r is the length of the common normal, and α is the angle around the common normal between the previous and current z-axis. We then used the Using the DH parameters, we calculate the homogenous transformation matrix, and expanded it for each link. Once simplified (when either cos(𝜋/2) or sin(0) == 0), we are left with three unique matrices. These were implemented in the algorithm in the form of , , and .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Link number | d (meters) | (radians) | r (meters) | (radians) |
| L1 | 4.0 | 𝜋/2 | 0 | 𝜋/2 |
| L3 | 0 | 𝜋/2 | 3.2 | 𝜋/2 |
| L4 | 0 |  | 2.8 | 0 |

:

|  |  |  |
| --- | --- | --- |
| Comparison number | Actual Position [x, y, z] | Calculated Position [x, y, z] |
| 1 | (7.18, 16.62, 10.75) | (11.03, 13.31, 0.98) |
| 2 | (7.18, 17.27, 11.78) | (15.29, 15.94, 6.42) |
| 3 | (7.78, 16.59, 10.59) | (10.49, 18.95, 6.72) |
| 4 | (10.68, 16.76, 11.00) | (12.23, 17.15, 1.41) |
| 5 | (13.00, 16.65, 11.51) | (14.10, 16.13, 8.42) |
| 6 | (13.42, 16.03, 12.09) | (8.14, 17.18, 1.22) |
| 7 | (10.76, 11.49, 12.02) | (11.34, 15.21, 7.03) |
| 8 | (10.76, 14.79, 12.04) | (5.88, 19.94, 3.89) |
| 9 | (13.77, 14.05, 12.61) | (13.65, 13.69, 6.78) |
| 10 | (10.79, 19.31, 10.72) | (14.25, 15.78, 0.17) |

*3.2: Inverse Kinematics*

Get the desired positions based on the trajectory specified in target.py. Using the joint angle estimates derived from vision\_2.py and the Jacobian in task 3.1, we computed the pseudo-inverse of the Jacobian We then followed the method in Lab 3 to derive the new joint velocities from the Jacobian inverse and the task space error.