IVR Assignment

Maksymilian Mozolewski & Martin Lewis

November 2020

Division of Labour

- Section 2 - Martin
- Section 3 - Maks
- Section 4 - Maks

Github

Link to the github repository: https://github.com/martin-lewis/ivr_assignment

2.1

The first part of this section was straight forward calculating the position of the angles based of the sinusoidal positions. This achieved with rospy's get_time() function and sin. This was then published to the topics to get the robot moving.

The next section is getting the 3D coordinates from the 2 2D images. This was done initially with a naive system that assumed that the camera was always orthogonal, which it isn't. This worked fine for most cases except the extreme cases where the robot arm would point directly into a camera. This was replaced by a more complex system considering the cameras to be simple pinhole cameras and reversing the projection into the 2D image back into the 3D world coordinates. Then the position of the object in both cameras is considered together to find the height.

Finally having got the 3D world coordinates then the angles had to be found. We tried a few different methods, optimisation, projections but finally settling for using trigonometry. Using atan2 to find the values of joint 2 and 3. Joint 2 is found with atan on the y and z components of the vector between the blue and green blobs. Then the vector is rotated by the found angle and then atan is run on the x and z components for joint3. This does however mean that the uncertainly and error in the angle of joint 2 is passed to joint3.

You can see the two example sections of an rqt_plot graph from joints 2 and 3 in figures 1 and 2.

Joint 4 however is calculated by a projection. This means its not reliant on the first two angles. It works out the angle between the vector between the green and red blobs and the vector between the blue and green ones.

2.2

The target is distinguished first by a threshold over the colour orange. Then the cv2 match template function is applied to it using two templates (template-box.png and template-sphere-png both are in root folder of the repository) this returns a position in the image supplied to it. This is then turned into 3D world coordinates by the system discussed in 2.1. You can see a graph showing the results in figure 4. The major issue is that of the z value, it is frequently not quite correct, it usually in the correct ball park but not as accurate at the other two. X and Y however are very accurate for the vast majority of the time.

I suspect that the source of error comes mainly from the fact that the target is usually high up in the cameras view and that this means it's at more of an angle reducing the accuracy of the algorithm in working out its Z coordinate.

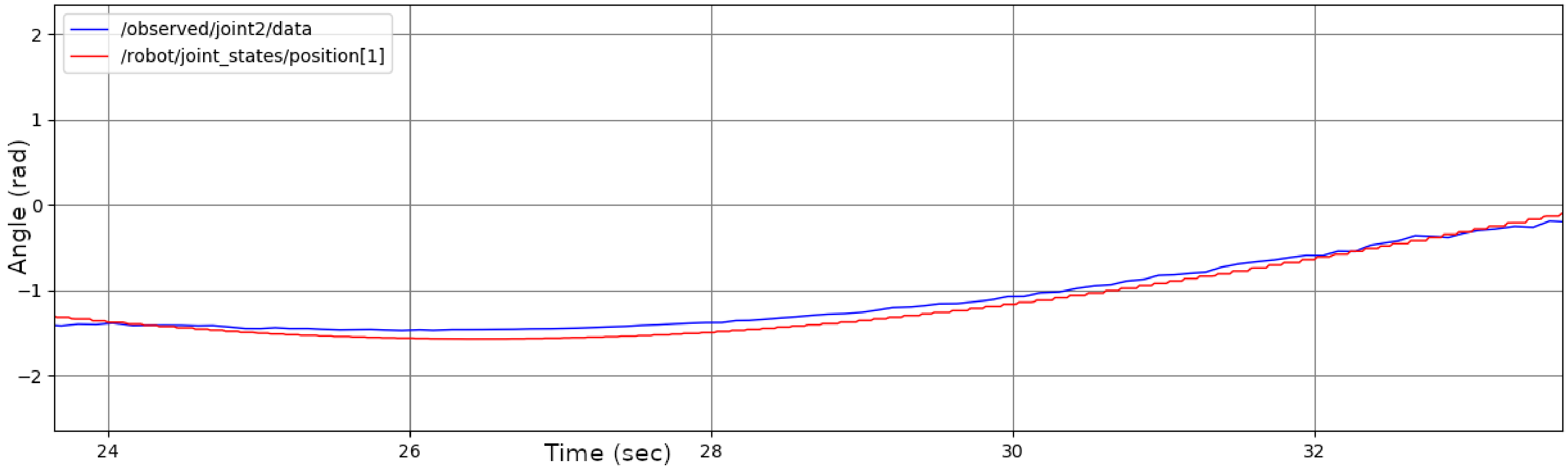

Figure 1: Joint2 Angle

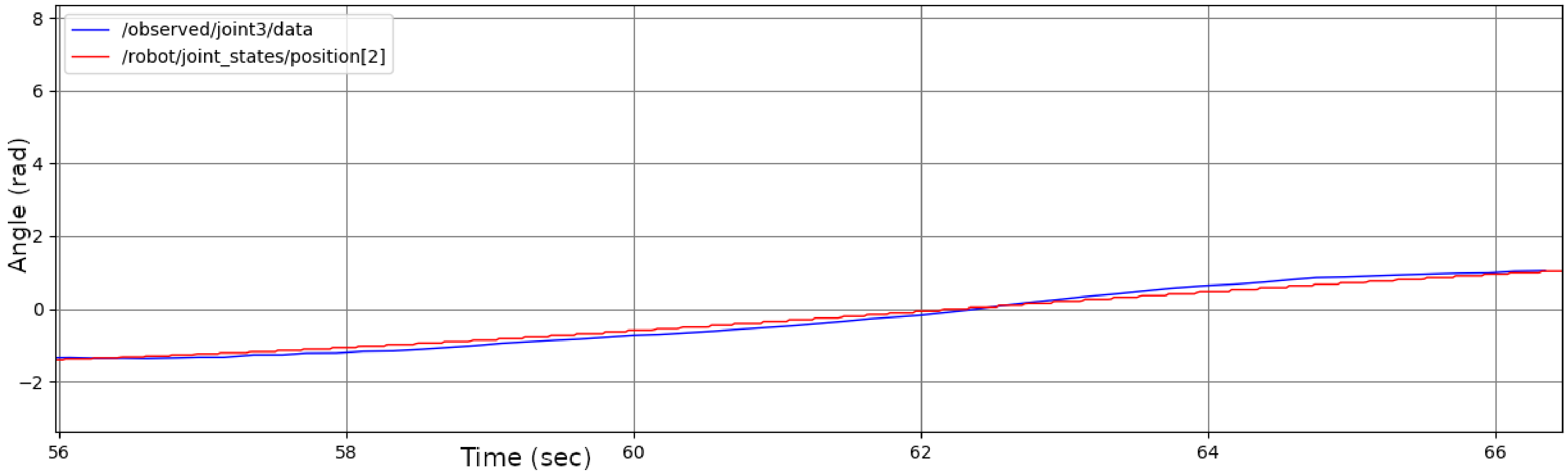

Figure 2: Joint3 Angle

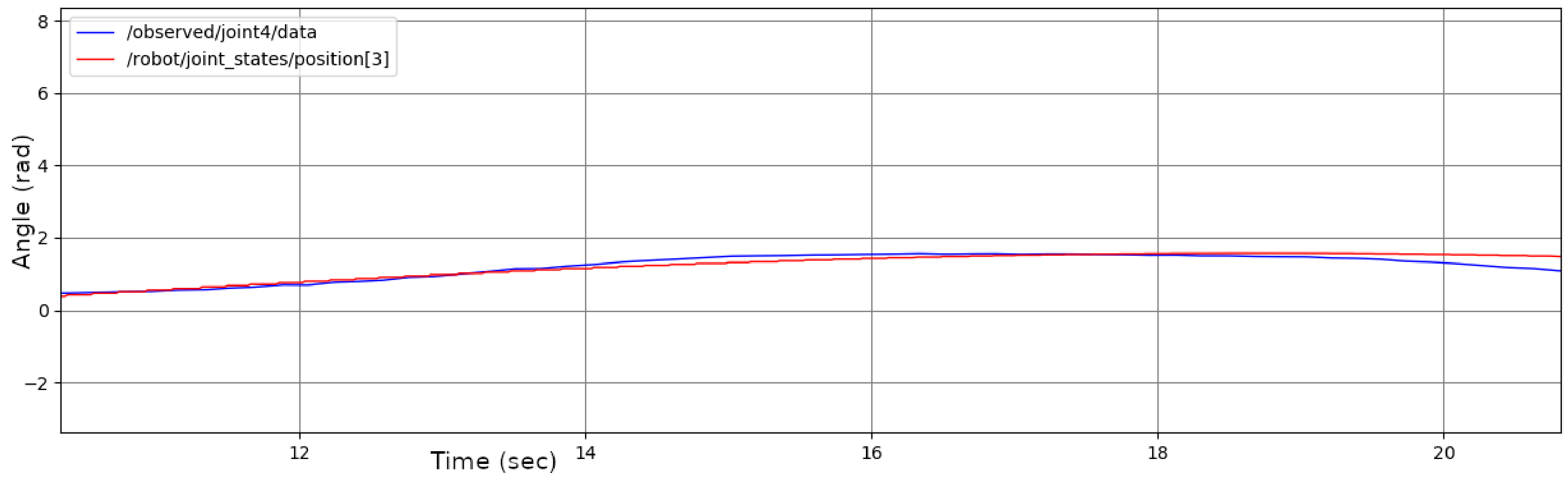

Figure 3: Joint4 Angle

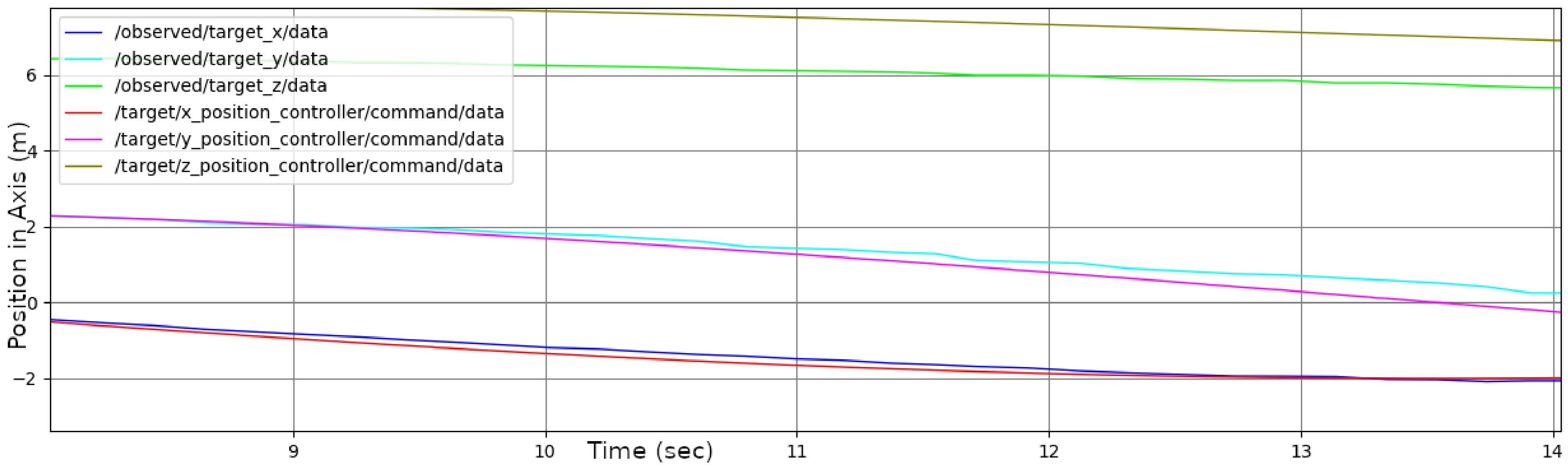

Figure 4: Target Position (/observed/...) and actual position (/target/...)

3.1 Forward Kinematics

Presentation of the forward kinematics equation

The forward kinematics, was derived using d-h parameters and a sympy script which parsed it to avoid any mathematical errors (The said script is available in the repository). The final forward kinematics translation component looks like the following:

$$FK(q) = \begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} = \begin{pmatrix} 3\left(S_{q_1} S_{q_2} C_{q_3} + S_{q_3} C_{q_1}\right) C_{q_4} + 3.5 S_{q_1} S_{q_2} C_{q_3} + 3 S_{q_1} S_{q_4} C_{q_2} + 3.5 S_{q_3} C_{q_1} \\ 3\left(S_{q_1} S_{q_3} - S_{q_2} C_{q_1} C_{q_3}\right) C_{q_4} + 3.5 S_{q_1} S_{q_3} - 3.5 S_{q_2} C_{q_1} C_{q_3} - 3 S_{q_4} C_{q_1} C_{q_2} \\ -3 S_{q_2} S_{q_4} + 3 C_{q_2} C_{q_3} C_{q_4} + 3.5 C_{q_2} C_{q_3} + 2.5 \end{pmatrix}$$

Comparing forward kinematics and vision algorithm's estimations of end-effector positions

The robot was set to the different joint positions given in figure below using a script which automatically set the robot's positions, waited for the arm to reach the target positions (3 seconds between each angle) and then stored the different predictions to a .tex file automatically.

This was to avoid any human error and image synchronisation with the robot issues. In the far column euclidian distance is used to compare the different estimations. We can see that all the estimations are between 0.5 and 0.75 meters off each other, this is quite close however not ideal. The highest differences appear for angles which direct the end effector closer or further to only one of the cameras, the source of error here is clearly not the forward kinematics equation, since those are the configurations which are the most problematic for vision. The sources of error here are:

- Obfuscation of the end effector by the robot. Both partial and full obfuscation causes: a shift in the CoM of the blobs, and loss of accurate information respectively in which case position data is extrapolated via average velocity of the past 5 measurements. This causes disparity between the vision and FK estimations
- Lighting variation, some of the blobs will sometimes be shifted very slightly when the colours become more shaded, this adds to the error and is likely the reason a constant error is present (this could be fixed by using the HSI colour space for thresholding instead)
- The shaking of the robot due to movement. Sometimes the robot will oscillate after performing a motion meaning that the values are slightly offset to the predicted FK positions, since the FK assumes a fixed frame at the bottom - but the base in the robot can move, causing the vision estimates to become more true than the FK estimates

q1(rad)	q2(rad)	q3(rad)	q4(rad)	Vision EFPos	FK EFPos	Euclidian Distance(m)
0.50	-0.50	0.50	-0.50	[0.672 4.907 5.993]	[0.738 4.782 6.534]	0.56
-0.50	0.50	-0.50	0.50	[-4.542 -2.081 5.983]	[-4.422 -1.962 6.534]	0.58
0.50	0.50	0.50	0.50	[4.511 -2.067 5.977]	[4.422 -1.962 6.534]	0.57
-0.50	-0.50	-0.50	-0.50	[-0.768 4.882 6.004]	[-0.738 4.782 6.534]	0.54
1.00	0.50	-1.00	0.50	[-0.4 -6.1 4.117]	[-0.389 -5.883 4.718]	0.64
0.50	1.00	-1.00	0.50	[-2.896 -5.787 2.423]	[-2.819 -5.603 3.08]	0.69
1.57	1.57	0.10	0.10	[6.58 0.652 1.561]	[6.453 0.647 2.2]	0.65
3.14	1.57	0.10	0.10	[-0.925 6.674 1.59]	[-0.647 6.453 2.2]	0.71
-1.57	1.57	1.00	0.50	[-3.398 -5.339 0.359]	[-3.314 -5.161 1.062]	0.73
-3.14	1.57	0.10	0.10	[-0.925 6.674 1.59]	[-0.647 6.453 2.2]	0.71

Figure 5: The End Effector Position estimates at the different joint configurations (q1,q2,q3,q4) via FK and Vision

3.2 Closed-Loop Control

Presentation of the velocity kinematics calculation

The Jacobian was again calculated using SymPy to avoid errors, the resulting matrix is quite large and so we will present it split by columns:

$$\mathrm{J(q)}_{*,1} = \begin{bmatrix} \left(-3S_{(q_1)}S_{(q_3)} + 3S_{(q_2)}C_{(q_1)}C_{(q_3)}\right)C_{(q_4)} - 3.5S_{(q_1)}S_{(q_3)} + 3.5S_{(q_2)}C_{(q_1)}C_{(q_3)} + 3S_{(q_4)}C_{(q_1)}C_{(q_2)} \\ \left(3S_{(q_1)}S_{(q_2)}C_{(q_3)} + 3S_{(q_3)}C_{(q_1)}\right)C_{(q_4)} + 3.5S_{(q_1)}S_{(q_2)}C_{(q_3)} + 3S_{(q_1)}S_{(q_4)}C_{(q_2)} + 3.5S_{(q_3)}C_{(q_1)} \\ 0 \end{bmatrix}$$

$$\mathrm{J(q)}_{*,2} = \begin{bmatrix} -3S_{(q_1)}S_{(q_2)}S_{(q_4)} + 3S_{(q_1)}C_{(q_2)}C_{(q_3)}C_{(q_4)} + 3.5S_{(q_1)}C_{(q_2)}C_{(q_3)} \\ 3S_{(q_2)}S_{(q_4)}C_{(q_1)} - 3C_{(q_1)}C_{(q_2)}C_{(q_3)}C_{(q_4)} - 3.5C_{(q_1)}C_{(q_2)}C_{(q_3)} \\ -3S_{(q_2)}C_{(q_3)}C_{(q_4)} - 3.5S_{(q_2)}C_{(q_3)} - 3S_{(q_4)}C_{(q_2)} \end{bmatrix}$$

$$\mathrm{J(q)}_{*,3} = \begin{bmatrix} \left(-3S_{(q_1)}S_{(q_2)}S_{(q_3)} + 3C_{(q_1)}C_{(q_3)}\right)C_{(q_4)} - 3.5S_{(q_1)}S_{(q_2)}S_{(q_3)} + 3.5C_{(q_1)}C_{(q_3)} \\ \left(3S_{(q_1)}C_{(q_3)} + 3S_{(q_2)}S_{(q_3)}C_{(q_1)}\right)C_{(q_4)} + 3.5S_{(q_1)}C_{(q_3)} + 3.5S_{(q_2)}S_{(q_3)}C_{(q_1)} \\ -3S_{(q_3)}C_{(q_2)}C_{(q_4)} - 3.5S_{(q_3)}C_{(q_2)} \end{bmatrix}$$

$$\mathrm{J(q)}_{*,4} = \begin{bmatrix} -\left(3S_{(q_1)}S_{(q_2)}C_{(q_3)} + 3S_{(q_3)}C_{(q_1)}\right)S_{(q_4)} + 3S_{(q_1)}C_{(q_2)}C_{(q_4)} \\ -\left(3S_{(q_1)}S_{(q_3)} - 3S_{(q_2)}C_{(q_1)}C_{(q_3)}\right)S_{(q_4)} - 3C_{(q_1)}C_{(q_2)}C_{(q_4)} \\ -3S_{(q_2)}C_{(q_4)} - 3S_{(q_4)}C_{(q_2)}C_{(q_3)} \end{bmatrix}$$

The algorithm deployed used the damped pseudo inverse of the Jacobian matrix. Only the Proportional and Derivative error components were used to direct the change in joint values at each iteration of the loop. The change in q (or angular velocity of q) at each iteration can be defined as follows:

$$\dot{q} = J^{*}(q)(K_d\dot{x}_d + K_p x_{\Delta})$$

where $J_d^*(q)$ is the damped pseudo inverse defined as:

$$J^*(q) = J^T(JJ^T + k^2\mathbf{I})^{-1}$$

x_Δ is the difference between the desired and current end effector position. $\dot{x}_d$ is the derivative of the end effector position and K_d,K_p are the constants used to control the size of each step in relation to the derivative and proportional errors at each step respectively.

Evaluation of control loop

(graphs)

4.2 Null-space Control

Algorithm

The only amendment to the previous algorithm needed to allow for a secondary task maximisation, is the addition of the null-space projection term:

$$\dot{q} = J^*(q)(K_d\dot{x}_d + K_p x_\Delta) + (\mathbf{I} - J^* J)\dot{q}_0$$

where $\dot{q}_0$ is the derivative of the "box" avoidance function w(q) defined as:

$$w(q) = ||FK(q) - b||$$

with q,b here standing for the estimate joint configuration and position of the obstacle (box) respectively at the current iteration of the loop. This change results in the required avoidance behaviour.

Plots

(graphs)