Pose difference estimation in cartesian impedance control

Asked 2 months ago Active 1 month ago Viewed 52 times

3

1

I am trying to implement a cartesian impedance controller in simulation (gazebo + KDL). The method which I am using is the following:

- Compute the EE pose from KDL's forward kinematics for the position
- Compute the EE twist from KDL's forward kinematics for the velocity
- Compute the Jacobian (which I am not sure how to figure out if it's the correct one - see later the comment about the rotation error) from the base frame to the EE frame
- Compute the difference betweent the EE's pose and the desired pose
- Multiply the difference with the Cartesian stiffness matrix
- Muttiply the velocity of the EE with the Cartesian damping matrix
- Sum the stiffness, damping
- Multiply the sum with the transpose of the Jacobian to go from Cartesian to joint space
- Add gravity compensation terms (computed by KDL)
- Apply the computed torque on the joints

This whole method is summarized here:

$$u = g(q) + J_a(q)[K_m(x_d - x) - D_m\dot{x}]$$

Note, for my use case (contactless, stable target position) the impedance controller reduces to PD + gravity compensation control.

The whole method seems to work, but I am running into the following issues:

- The target position takes a lot of time to reach and in the meantime there are some weird oscillations (see video here https://drive.google.com/file/d/1tC20Jf24BbdLCqRbOItIvdZIkqAq1Gfn/view?usp=sharing)
- I am very unsure of how to find the difference between the rotational part of the EE pose and target pose. For now I am using quaternions but I am not sure that the way that I am computing the difference is correct. To compute the rotation error I multiply the inverse of the EE pose quaternion with the target pose quaternion. Does that sound right? How can I make sure that the Jacobian uses the same representation for the rotational part? In general, any advice on how to properly compute the error between two poses?

Thanks a lot for the help!

control simulation gazebo

Share Improve this question Follow

asked Jul 23 at 8:45

Manos Agelidis
31 3

1 Welcome to *Robotics*, Manos Agelidis! Great first question, and I'm looking forward to seeing some answers regarding how to handle the quaternions. I watched the video and I'm not sure what exactly could be causing the slowdown issue, so I'd highly recommend *plotting* your terms. Having a way to visualize all your parameters should help you quickly narrow down potential root causes. Is your reference trajectory moving slowly at the end? Is the reference moving at the same speed but your reference tracking is over damped? – Chuck Jul 23 at 12:56

Thanks for the tip, will try plotting! – Manos Agelidis Jul 25 at 7:04

1 Answer

Active | Oldest | Votes

2

1. Weird motion

I'm pretty sure it happen because you make wrong implementation on orientation control.

2. Orientation of end-effector

There are 4 main representation of end-effector orientation which is axis angle, rpy (expanded into 6 types), euler angle (expanded into 6 types) and unit quarternion. Normal jacobian derived

freshly from forward kinematic matrix is for axis angle representation, a basic 3x3 matrix representation of orientation located on top left of forward kinematic matrix.

$$FK(\theta) = \begin{bmatrix} R_{3\times3} & D_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}$$

For axis angle representation error computation, with

Robot orientation (each letter represent column) :

$$R_e = [\, n_e \quad s_e \quad a_e \,]$$

Desired orientation :

$$R_d = [\, n_d \quad s_d \quad a_d \,]$$

Error would be :

$$e = \frac{1}{2}(n_e \times n_d + s_e \times s_d + o_e \times o_d)$$

If you wanna use other representation such as quarternion the error computation would be different and you would need analytical jacobian Ja.

$$J_a = T \times J$$

where T is transformation matrix. For more detail you can look at "Robotics motion, planning and control" by B. Siciliano sect 3.7.3 for orientation error and sect 3.6 for analytical jacobian. Or code implementation by Peter Corke on his github here

3. Quarternion error

Your method error computation already correct,

$$e = Q_d * Q_e^{-1}$$

Share Improve this answer Follow

edited Jul 24 at 11:36

answered Jul 24 at 5:46

Albert H M
589 4 15

Great answer, thanks! – Manos Agelidis Jul 25 at 7:04