Assignment 01: Comparison between different controllers

Elisa Alboni* - elisa.alboni@unitn.it

October 2021

1 Description

The goals of this assignment are:

- implementing different controllers: operational space control and impedance control
- comparing the performance of different versions of the impedance controller to stabilize a desired position of the end effector of a UR5 robot with different Coulomb friction values
- comparing the performance of the controllers using different feedback gains and frequency values of the reference trajectory on the UR5 robot

2 Submission procedure

You are encouraged to work on the assignments in groups of 2 people. **If you have a good reason to work alone, then you can do it, but this has to be previously validated by one of the instructors**. Groups of more than 2 people are not allowed. The mark of each assignment contributes to 10% of you final mark for the class (i.e. 3 points out of 30).

When you are done with the assignment, please submit a single compressed file (e.g., zip). **The file name should contain the surnames of the group members**, and it must contain:

- A pdf file with the answers to the questions, the **names and ID number** of the group members; you are encouraged to include plots and/or numerical values obtained through simulations to support your answers. **This pdf does not need to be long. One or two pages of text should be enough to answer the questions. You can then add other pages for plots and tables.**
- The complete orc folder containing all the python code that you have developed.

If you are working in a group (i.e., 2 people) only one of you has to submit.

Submitting the pdf file without the code is not allowed and would result in zero points. Your code should be consistent with your answers (i.e. it should be possible to produce the results that motivated your answers using the code that you submitted). If your code does not even run, then your mark will be zero, so make sure to submit a correct code.

*Optimization and Learning for Robot Control, Industrial Engineering Department, University of Trento.

3 Controllers

In this section we will briefly revise the two controllers to be implemented: Operational Space Control (OSC) and Impedance Control (IC).

3.1 Operational Space Control (OSC)

Operational-Space control is a well-known approach for controlling the position of the end effector of a robot manipulator. Given the standard joint-space dynamics of a robot manipulator:

$$M\ddot{q} + h = \tau \tag{1}$$

We can write the operational-space dynamics:

$$\Lambda\ddot{x} + \mu = J^{\top\dagger}\tau = f \tag{2}$$

where $\Lambda \triangleq \left(JM^{-1}J^{\top}\right)^{-1}$ is the operational-space inertia of the end effector, $\mu \triangleq \Lambda(JM^{-1}h - \dot{J}\dot{q})$ are the operational-space bias forces (due to gravity, Coriolis and centrifugal forces), $J^{\top\dagger} \triangleq \Lambda JM^{-1}$ is a pseudo-inverse of $J^{\top}$ using M^{-1} as weight matrix, and finally f is our new control input, defined via $\tau = J^{\top}f$.

Given a desired acceleration $\ddot{x}^d$, typically computed with a linear PD control law ($\ddot{x}^d \triangleq \ddot{x}^{ref} + K_p(x^{ref} - x) + K_d(\dot{x}^{ref} - \dot{x})$, where K_p and K_d have diagonal elements respectively kp and kd), the corresponding desired value of f is simply computed as:

$$f^d = \Lambda\ddot{x}^d + \mu \tag{3}$$

The corresponding joint torques are given by $\tau = J^{\top}f^d$. Finally, we can add to τ another control signal to stabilize a particular joint configuration without affecting the operational-space acceleration:

$$\tau = J^{\top}f^d + (I - J^{\top}J^{\top\dagger})\tau_0 \tag{4}$$

where $\tau_0 \triangleq M\ddot{q}^{d_{pos}} + h$, $\ddot{q}^{d_{pos}} \triangleq K_{pj}(q^{d_{pos}} - q) - K_{dj}\dot{q}$ and K_{pj} and K_{dj} have diagonal elements k_{pj} and k_{dj}, respectively. Let's choose to stabilize the initial joint configuration, namely $q^{d_{pos}} \triangleq q^0$.

3.2 Impedance Control (IC)

Impedance control tries to make the system behave as a linear impedance:

$$\Lambda\ddot{e} + B\dot{e} + Ke = f_{ext} \tag{5}$$

where $e \triangleq x^{ref} - x$ is the tracking error, f_{ext} are the external forces applied to the end effector, and Λ, B and K are the operational-space inertia, damping and stiffness matrices. The following control law achieves this dynamical behaviour of the end effector:

$$\tau = J^T\left(Ke + B\dot{e} + \mu\right) \tag{6}$$

where μ are the biased forces (due to gravity, Coriolis and centrifugal forces). To avoid the computation of μ, we can use the simplified version of IC:

$$\tau = h + J^T\left(Ke + B\dot{e}\right) \tag{7}$$

Similarly to Operational Space Control, an additional control signal can be added to stabilize the joint space motion.We can modify the simplified IC as follows:

$$\tau = h + J^T \left(K(x^{ref} - x) + B(\dot{x}^{ref} - \dot{x}) \right) + \left(I - J^\top J^{\top\dagger} \right) \tau_0 \tag{8}$$

Note that in this case τ_0 does not need to include the bias forces h, which are already compensated, so it is simply defined as $\tau_0 \triangleq M\ddot{q}^{d_{pos}}$. To add this postural task to the exact version of IC instead, the bias forces h must be included in τ_0

4 Tests

The template code for this part of the assignment is located in:

`orc/reactive_control/A1_template.py`

This file already contains all the code for comparing the different versions of IC and OSC and IC. The only parts that need to be implemented are the control laws inside the for loop:

1. `IC_O_simpl` = Impedance Control (simplified version)
2. `IC_O_simpl_post` = Impedance Control (simplified version with postural task)
3. `IC_O` = Impedance Control (exact version)
4. `IC_O_post` = Impedance Control (exact version + postural task)
5. `OSC` = Operational Space Control (postural task)
6. `IC` = Impedance Control (simplified version + postural task)

When you are done with one controller, if you want to test it, you can run the script commenting the tests that use the other controllers not implemented yet.

In the first part, the four versions of IC are compared using two Coulomb friction values (0 and 2% of the maximum torque at each joint) to stabilize a desired position, `x_ref_O`. The simulations are executed as specified by these lines in the script (no friction)

```
tests = []
tests += ['controller':  'IC_O_simpl', 'kp':  250, 'friction':  0]
tests += ['controller':  'IC_O_simpl_post', 'kp':  250, 'friction':  0]
tests += ['controller':  'IC_O', 'kp':  250, 'friction':  0]
tests += ['controller':  'IC_O_post', 'kp':  250, 'friction':  0]
```

and by these lines in the script (friction):

```
tests = []
tests += ['controller':  'IC_O_simpl', 'kp':  250, 'friction':  2]
tests += ['controller':  'IC_O_simpl_post', 'kp':  250, 'friction':  2]
tests += ['controller':  'IC_O', 'kp':  250, 'friction':  2]
tests += ['controller':  'IC_O_post', 'kp':  250, 'friction':  2]
```

Note: also the frequency is specified but not used.

Then, OSC and IC are compared using two frequency values of the sinusoidal reference trajectory to track (set to 1 the flag `TRACK_TRAJ` in the configuration file):

```
tests = []
tests += ['controller':  'OSC', 'kp':  100, 'frequency':  np.array([1.0, 1.0, 0.3])]
tests += ['controller':  'IC', 'kp':  100, 'frequency':  np.array([1.0, 1.0, 0.3])]
```
and
```
tests = []
tests += ['controller':  'OSC', 'kp':  100, 'frequency':  3*np.array([1.0, 1.0, 0.3])]
tests += ['controller':  'IC', 'kp':  100, 'frequency':  3*np.array([1.0, 1.0, 0.3])]
```
Note: also the friction is specified and must not be changed.

In both cases, only the proportional gain k_p is specified because the derivative gain is always chosen as $2\sqrt{k_p}$. Please do not change any parameter in the configuration file `A1_conf.py` if not explicitly asked, except for `use_viewer` and `simulate_real_time` that allow you to enable/disable the viewer and run the simulation either in real time or as fast as possible. The script prints and plots the average tracking error obtained with each simulation, which can be used to compare the performance of the controllers.

Try to answer the following questions:

If you want to stabilize a desired position with different versions of IC,

1. What can you say about the relative performance of the different implementations of IC in the different settings? What is the effect of friction on the performance of the different versions of the IC controller? Report the values of the tracking errors that you got in simulation.
2. In one case, a controller fails to stabilize the end effector. In your opinion, why does it happen and why do the other controllers manage to stabilze the desired position?

If instead of a desired position, you want to stabilize a reference trajectory with IC and OSC (set `TRACK_TRAJ`=1 in the configuration file),

1. What can you say about the relative performance of OSC and IC in the different settings? Which controller performed best at higher frequency of the reference trajectory? How does the relative performance of the controllers varies considering the two frequency values? Report the values of the tracking errors that you got in simulation.
2. Which parameters of the control laws could you tune to improve the controllers performances? Re-tune the controllers according to your ideas and report the tracking error obtained.
3. Test the robustness of the controllers to modelling errors by setting to 1 the flag `randomize_robot_model` in the configuration file. How did the performance of OSC and IC change?