Assignment 01: Comparison between different controllers

Gianluigi Grandesso* - gianluigi.grandesso@unitn.it

October 2021

1 Description

The goals of this assignment are:

- implementing different controllers: operational space control, impedance control and inverse kinematics - inverse dynamics control
- 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-based Robot Control, Industrial Engineering Department, University of Trento.

3 Controllers

In this section we will briefly revise the three controllers to be implemented: Operational Space Control (OSC), Impedance Control (IC) and Inverse Kinematics - Inverse Dynamics Control (IKID).

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 approximately achieves the desired dynamical behavior of the end-effector:

$$\tau = h + J^T\left(K(x^{ref} - x) + B(\dot{x}^{ref} - \dot{x})\right) \tag{6}$$

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:

$$\tau = h + J^T\left(K(x^{ref} - x) + B(\dot{x}^{ref} - \dot{x})\right) + \left(I - J^{\top}J^{\top\dagger}\right)\tau_0 \tag{7}$$

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 $\tau_0 \triangleq M\ddot{q}^{d_{pos}}$.

3.3 Inverse Kinematics - Inverse Dynamics Control

This last controller first retrieves the desired joint accelerations $\ddot{q}^d$ by inverting the kinematics:

$$\ddot{q}^d = J^+(\ddot{x}^d - \dot{J}\dot{q}) \tag{8}$$

where J^+ is the right Moore-Penrose pseudo-inverse of J defined as:

$$J^+ \triangleq J^\top (JJ^\top)^{-1} \tag{9}$$

As with the other two controllers, we can take advantage of the null space projector $(I - J^+J)$ to stabilize also the initial joint configuration q_0, this time working directly in the joint space:

$$\ddot{q}^d = J^+(\ddot{x}^d - \dot{J}\dot{q}) + \left(I - J^+J\right)\ddot{q}^{d_{pos}} \tag{10}$$

Note that also in this case the additional control signal does not include the bias forces h because already compensated in the final control law, which is obtained by simply inverting the dynamics:

$$\tau = M\ddot{q}^d + h \tag{11}$$

4 Tests

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

```
code/orc/01_assignment/OSC_vs_IC_vs_IKID_template.py
```

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

1. `OSC` = Operational Space Control
2. `IC` = Impedance Control (simplified version)
3. `IKID` = Inverse Kinematics - Inverse Dynamics Control

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. The three controllers are compared using two frequency values of the sinusoidal reference trajectory to track, and two gains for each frequency value. Therefore, at each run, 16 simulations are executed, as specified by these lines in the script:

```
tests = []
tests += ['controller': 'OSC', 'kp': 50, 'frequency': np.array([1.0, 1.0, 0.3])]
tests += ['controller': 'IC', 'kp': 50, 'frequency': np.array([1.0, 1.0, 0.3])]
tests += ['controller': 'IKID', 'kp': 50, 'frequency': np.array([1.0, 1.0, 0.3])]
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])]
tests += ['controller': 'IKID', 'kp': 100, 'frequency': np.array([1.0, 1.0, 0.3])]
tests += ['controller':  'OSC', 'kp':  50, 'frequency':  np.array([2.0, 2.0, 0.6])]
tests += ['controller':  'IC', 'kp':  50, 'frequency':  np.array([2.0, 2.0, 0.6])]
tests += ['controller':  'IKID', 'kp':  50, 'frequency':  np.array([2.0, 2.0, 0.6])]
tests += ['controller':  'OSC', 'kp':  100, 'frequency':  np.array([2.0, 2.0, 0.6])]
tests += ['controller':  'IC', 'kp':  100, 'frequency':  np.array([2.0, 2.0, 0.6])]
tests += ['controller':  'IKID', 'kp':  100, 'frequency':  np.array([2.0, 2.0, 0.6])]
```

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 `OSC_vs_IC_vs_IKID_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. In the viewer you can see the green sphere, representing the end effector, that tries to track the reference trajectory executed by the red sphere. 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:

1. What can you say about the relative performance of OSC, IC and IKID in the different settings? Which controller performed best at higher frequency of the reference trajectory? How does the relative performance of the controllers vary considering the two frequency values? Report the values of the tracking errors that you got in simulation.
2. Set to 1 the flag `randomize_robot_model` in the configuration file and re-run the script. This flag enables the random modification of the inertial parameters of the simulated robot, making it different (at most 40%) from the model used by the controllers. How did the performance of OSC, IC and IKID change? Which of the three controllers was most robust to the introduced modeling errors? [1]
3. Visualize the motions of the robot in the viewer (`use_viewer`= 1 and `simulate_real_time`= 1) and explain why in some cases (*e.g.* `test = ['controller': 'IKID', 'kp': 50, 'frequency': np.array([2.0, 2.0, 0.6])]`) the robot makes some abrupt moves. Which parameters of the control laws could you tune to avoid this behavior? Why? [Hint: reason about what does not affect the operational-space dynamics]
4. Try to tune the parameters as you suggested in the answer to the previous question. How do the results change in terms of mean tracking error? Why?
5. (OPTIONAL) Which control law would you choose to implement on a real robot? Why? And how could you try to improve its performance? Discuss different ways, pros and cons, possibly showing plots and tracking errors that you obtained by testing your ideas in simulation.

[1]Every time that you re-run the script you will get slightly different results because of the randomization of the robot model, so you should run the script a few times to avoid basing your answers on a particularly lucky/unlucky case.