LAB 3: Task Space Motion Control

Michele Focchi and Octavio Villarreal
Edited by Thiago Boaventura

The goals of this assignment are:

- To learn the basic procedure to design a motion controller in the *task* space for a manipulator in free motion (i.e. not in contact)
- To analyze the advantages/disadvantages of decentralized/ centralized approaches (i.e. feedback linearization)
- To implement the control of interaction with impedance control in task space
- *Optional:* to implement the control of the orientation using angle-axis representation

Tools that we are going to be using in this lab (all open-source):

- Python programming (2.7)[1]
- Robot Operating System (ROS)[2]
- Pinocchio Library for Rigid Body Dynamics Algorithms[3]
- Locosim[4]

In this assignment we will design the reference trajectory and close the loop at the end-effector point. We will have to deal with dynamic coupling and how to appropriately solve the indeterminacy, in the case the robot is redundant with respect to the task dimension. Initially, we will design some reference trajectories to track, defined at the joint level. Then, we will design several control laws with increasing complexity evaluating their impact on the accuracy and the tracking errors of the joint.

0 Preliminaries

To test the RVIZ visualization and have a grasp on the kinematic of the UR5 robot, try to move the joints with the sliders, running:

```
roslaunch visualize.launch robot_name:=ur5 test_joints:=true
```

[1] https://docs.python.org/2.7/
[2] https://www.ros.org/
[3] https://github.com/stack-of-tasks/pinocchio
[4] https://github.com/mfocchi/locosim

1 Decentralized task space control

1.1 - Generate Sinusoidal Reference: Generate a sinusoidal reference for the X direction of the end-effector with 0.1 m amplitude and 1.5 Hz frequency, starting from the initial configurations p_0. Keep the other directions Y and Z at 0. After 4.0 s stop the sine and give a constant reference during 1.0 s. Note that you also need to generate consistent references for velocity and acceleration.

1.2 - Step Reference generation: Generate a 0.1 m step reference at $t = 2.0s$ for for the Z direction (keep the other directions with 0 reference), starting from the initial configurations p_0.

1.3 - Inverse kinematics approach (Optional for laude): Implement the end-effector tracking using inverse kinematics implemented in L1 lab, to map the end-effector position to joint space and the other differential mappings for velocity and acceleration, i.e.:

$$\begin{cases} q^d = IK(p^d) \\ \dot{q}^d = J^{\#}\dot{p}^d \\ \ddot{q}^d = J^{\#}(\ddot{p}^d - \dot{J}\dot{q}) \end{cases}$$

Notice that since the task is three-dimensional, the Jacobian will not be square and you need to employ $J^{\#}$ rather J^{-1}. Use a standard PD control to close the loop at the joint level, with a feed-forward term. Evaluate the tracking of the end-effector.

1.4 Cartesian Space PD control: Implement a PD control law for the position of the end-effector (that's why the word Cartesian).

$$\tau_c = J^T \Big(K_x(p^d - p) + D_x(\dot{p}^d - \dot{p}) \Big) \tag{1}$$

Since we are considering only the position of the end-effector, you should use only the first 3 rows of the Jacobian $J \in \mathbb{R}^{6\times 6}$. Use as proportional gain the positive definite matrix $K_x = diag(1000, 1000, 1000)$ N/m and as the derivative gain $D_x = diag(300, 300, 300)$ Ns/m. Try to simulate and observe how the robot behaves without any *postural task* to solve the indeterminacy (we have a 3D task and 6 DoFs at the joints). Use the sinusoidal reference generated in (1) as input to the system. Plot the tracking error, see that there is a big steady-state error due to gravity in the Z direction but that there are also couplings that affect the tracking in the X, Y directions. Try to change the stiffness gain K_x in the X,Y directions and see that their tracking improves. See the joints are moving randomly because there is no control in the null space and we have 6 DoFs with a $3D$ task, and this motion affects the tracking of the end-effector.

1.5 - Cartesian Space PD control - postural task: Add a secondary postural task in the joint space with gains $K_q = 50I_{6\times6}$, $D_q = 10I_{6\times6}$ to attract the joints to a default posture q_0 in the joint space:

$$\tau_0 = K_q(q_0 - q) - D_q\dot{q} \tag{2}$$

$$N = I_{6\times6} - J^T J^{T\#} \tag{3}$$

then add the contributions of the two tasks. By using the Null-space projector N of $J^{T\#}$ you ensure that the postural task does not enter in conflict with the primary end-effector task:

$$\tau = \tau_c + N\tau_0 \tag{4}$$

Observe that the joints now are no longer wobbling around, and the tracking of the end-effector is no longer affected by the random motion of the joints. This is because this lives in the null-space of $J^{T\#}$ (that has the same dimension of the null-space of J), that is, the set of torques that do not affect the end-effector motion.

1.6 - Cartesian Space PD control + Gravity Compensation: Add a gravity compensation term computed in the task space $G = J^{-T}g$).

$$\tau_c = J^T\left(K_x(p^d - p) + D_x(\dot{p}^d - \dot{p}) + J^{-T}g\right) \tag{5}$$

Give the step reference generated in (2) as input to the system. Check there is no longer a tracking error at steady state but it still remains during the transient.

1.7 - Cartesian PD control + Gravity Compensation + Feed-Forward term: Add also a feed-forward acceleration action to the PD control. Use the the reflected inertia at the end-effector $\Lambda = (JM^{-1}J^T)^{-1}$ (note that we no longer use only the diagonal elements!):

$$\tau_c = J^T\left(\Lambda\ddot{p}^d + K_x(p^d - p) + D_x(\dot{p}^d - \dot{p}) + J^{-T}g\right) \tag{6}$$

Note that J should be full row rank otherwise you need to add a damping term to make $JM^{-1}J^T$ invertible. We employ the expression of $\Lambda = (JM^{-1}J^T)^{-1}$ because this is valid also for redundant robots, while $\Lambda = J^{-T}MJ^{-1}$ is only valid for nonredundant robots. Give the sinusoidal reference generated in (1). See that the tracking error is improved for a time-varying reference on the X direction, but still there are tracking errors on Y and

Z direction due to the inertial couplings and to the inconsistency of the initial state (for velocity).

2 Centralized task space control (Lecture H0)

Similarly to what we did in the joint space case, to effectively compensate for the inertial couplings, we will compensate them by linearizing the Cartesian dynamics of the robot (i.e. reflected at the end-effector) via a state feedback.

2.1 - Cartesian space inverse dynamics: Implement Cartesian space inverse dynamics with a postural task in the null space of the end-effector task, to attract the robot configuration q_0. Observe that the tracking is now almost perfect, there are no coupling between different directions anymore. Observe that if you reduce the gain, the tracking remains good despite the discontinuity at the end of the sinusoidal trajectory.

$$F^d = \ddot{p}^d + K_x(p^d - p) + D_x(\dot{p}^d - \dot{p}) \tag{7}$$
$$\mu(q, \dot{q}) = J^{T\#}h - \Lambda\dot{J}\dot{q} \tag{8}$$
$$\tau_c = J^T\left(\Lambda F^d + \mu(q, \dot{q})\right) \tag{9}$$

2.2 - Cartesian space inverse dynamics - simplified: Rather than compensate for the bias terms reflected at the end-effector $\mu(q, \dot{q})$, compensate for them at the joint space level (h) and observe the result is the same.

$$F^d = \ddot{p}^d + K_x(p^d - p) + D_x(\dot{p}^d - \dot{p}) \tag{10}$$
$$\tau_c = J^T\left(\Lambda F^d)\right) + h \tag{11}$$

The advantage of this (less elegant) approach is that it saves us the computation of the terms $\dot{J}$ and $J^{T\#}$.

2.3 - External Force: With constant reference x_0, apply an external force $F_{ext} = (0.0, 0.0, 200)N$ (i.e. on Z direction) at t $= 1$ s and check that, thanks to the inverse dynamics, only the end-effector position in the Z direction is affected, but not on X and Y. If you reduce by half the K_x gains you should see the deflection due to F_{ext} it doubles.

3 Orientation control (Optional)

Until now we only defined a 3D task for the end-effector position. Now we want also to control the orientation and define a full 6D task. The secondary postural task will no longer be necessary because the dimension of the task is the same as the number of joints

(i.e. dimension of the joint space). Note that you can control the orientation employing different parametrization of the orientation to compute the orientation error.

3.1 - PD control of end-effector position + orientation: Add orientation control of the end-effector (advice: the Jacobian is 6×6 now). Implement a PD control law and using the *angle-axis* representation to compute the orientation error ${}_{w}e_{o}$.

First, compose the rotation matrix that represents the desired orientation ${}_{w}R_{d} = [\hat{x}^{d}, \hat{y}^{d}, \hat{z}^{d}]$ with ($\hat{x}_{d}$ axis along $\hat{x}_{w}$, $\hat{y}_{d}$ axis along $-\hat{y}_{w}$, $\hat{z}_{d}$ axis along $-\hat{z}_{w}$). Then compute the rotation matrix that represents the relative rotation from the *actual* end-effector orientation ${}_{w}R_{ee}$ to the *desired* orientation ${}_{w}R_{d}$:

$$ {}_{ee}R_{d} = {}_{w}R_{ee}^{T}\,{}_{w}R_{d} \tag{12} $$

then, the orientation error ${}_{ee}e_{o}$ could be computed from the angle-axis associated to ${}_{ee}R_{d}$:

$$ {}_{ee}e_{o} = \delta\theta\hat{r} \tag{13} $$

where the unit axis $\hat{r}$ and the scalar $\delta\theta$ could be computed from the relative orientation matrix ${}_{ee}R_{d}$ (that we name R in this formula):

$$ \delta\theta = acos\left(\frac{R_{1,1} + R_{2,2} + R_{3,3} - 1}{2}\right) \tag{14} $$

$$ \begin{bmatrix} \hat{r}_{x} \\ \hat{r}_{y} \\ \hat{r}_{z} \end{bmatrix} = \frac{1}{2sin\delta\theta} \begin{bmatrix} R_{3,2} - R_{2,3} \\ R_{1,3} - R_{3,1} \\ R_{2,1} - R_{1,2} \end{bmatrix} \tag{15} $$

Note that the orientation error ${}_{ee}e_{o}$ it is expressed in the end-effector frame, we need to map it in the world frame to compute the moment because the jacobian J is expressed in the WF:

$$ e_{o} = {}_{w}R_{eeee}e_{o} \tag{16} $$

We can extract the angular velocity of the end-effector frame as the angular part of the 6D *twist* obtained as:

$$ \begin{bmatrix} \dot{p} \\ \omega \end{bmatrix} = J_{6\times6}\dot{q} \tag{17} $$

Now, we can write the PD law using the orientation proportional $K_{o} = I_{6\times6}800$ and

damping $D_o = I_{6\times6}30$ gains, setting the desired angular velocity to (0.0,0.0,0.0):

$$\Gamma^d = K_o e_o + D_o(\omega^d - \omega) \tag{18}$$

$$\tag{19}$$

Finally, we stack the orientation task Γ^d below the Cartesian task F^d (that was previously computed) to form the desired *wrench* W^d that we map to torques:

$$W^d = \begin{bmatrix} F^d \\ \Gamma^d \end{bmatrix} \qquad\qquad \tau = J^T W^d \tag{20}$$

Plot the orientation error and see that it is going to zero. Note that, since you used a PD control law, you will have tracking errors (both in the linear and in the angular coordinates) because of gravity and inertial couplings. Additionally, you will notice that increasing the damping gain $D_o = I_{6\times6}30$ beyond 30 makes the system unstable. We saw that the acos(.) function provides an output only in the $\Big[0, \pi\Big]$ range. Use the atan2(.) function to compute the $\delta\theta$ that provides output in $\Big[-\pi, \pi\Big]$.

$$\delta\theta = atan2\left(\sqrt{(R_{3,2} - R_{2,3})^2 + (R_{1,3} - R_{3,1})^2 + (R_{2,1} - R_{1,2})^2}, tr(R) - 1\right) \tag{21}$$

3.2 - PD control of end-effector position + orientation: singularity Now we want to change the orientation set-point ${}_wR_des$ at will. To do so is is convenient to use Euler Angles. Define the sequence (note it is not a vector) Φ^d of Euler Angles in ZYX convention that represent the desired orientation and map it into ${}_wR_des$ using the function `eul2Rot($\Phi^d$)`.

To check the tracking in terms of Euler Angles, you need to convert the rotation matrix ${}_wR_{ee}$ that represents the actual orientation into the correspondent Euler Angles using the dual function `rot2eul(${}_wR_{ee}$)`.

Set some values for the sequence Φ^d such that the representation is at singularity (i.e. $\theta = \pi/2$), and see that while the error orientation angle-axis is still continuous the plot of the Euler Angle representation suffers from discontinuities. This is one of the reasons for which we chose to close the loop on orientation error and not on Euler Angles for orientation control.

3.3 - PD control of end-effector position + orientation: sinusoidal reference Try to set a sinusoidal reference (which mimics a drilling operation) $2.2\pi sin(t)$ for the desired roll angle ψ^d and 0.0 for the desired pitch and yaw. Remember that you should also compute a consistent time-varying reference for the angular velocity ω_des in a PD control law. In this case, remember that you need to compute the derivatives of the Euler

Angles (Euler rates) and map them into ω via $\omega^d = T_\omega(\Phi^d)\dot{\Phi}^d$. Plot Euler Angle tracking.

3.4 - PD control of end-effector position + orientation: unwrapping Note that the roll signal switches between π and $-\pi$. You do not see this in the orientation error because it is always in the range π and $-\pi$. Implement an *unwrapping* algorithm to avoid this phenomena and get a continuous plot:

```
for i in range(3):
    rpy_unwrapped[i] = rpy[i];
    while (rpy_unwrapped[i] < rpy_old[i]  - math.pi):
        rpy_unwrapped[i] += 2*math.pi
    while (rpy_unwrapped[i] > rpy_old[i]  + math.pi):
        rpy_unwrapped[i] -= 2*math.pi
    rpy_old[i] = rpy_unwrapped[i]
```

Check at the end of this video what can happen to your robot if you do not do the unwrapping and you close the loop with Euler Angles to control the orientation!

3.5 - Using quaternions: Implement the computation of the orientation error with the quaternion representation and check that it gives the same result as closing the loop with the angle-axis representation.

3.6 - Implement the full (6D) task space inverse-dynamics: Implement the full inverse dynamics algorithm to remove the inertial/coriolis couplings and gravity for a 6D task. Observe that the Λ should be recomputed from the 6×6 Jacobian. Note that this matrix can become singular if the Jacobian is no longer full-row rank. Actually, notice that the rank of the Jacobian drops from 6 to 5 in the initial position q_0, therefore some mobility is lost at the very beginning.

$$J = \begin{bmatrix} -0.19145 & 0.27456 & -0.08306 & -0.08306 & 0.07223 & 0. \\ 0.5765 & 0. & 0. & 0. & 0. & 0. \\ 0. & -0.5765 & -0.34687 & 0.04538 - 0.03946 & 0. & \\ 0. & 0. & 0. & 0. & -0.47943 & 0. \\ 0. & 1. & 1. & 1. & 0. & 1. \\ 1. & 0. & 0. & 0. & -0.87758 & 0. \end{bmatrix} \tag{22}$$

How do you solve this? Try to set a threshold $\rho = 0.0001$ for the singular values in the SVD decomposition in the `np.linalg.pinv` function or, equivalently, implement a damping in the pseudo-inverse (check that they produce the same output).

$$J^{\#} = J^T \left(JJ^T + I_{3\times 3}\rho^2 \right)^{-1} \tag{23}$$

Remember that in this control law we need also to add the desired *acceleration* terms

(both linear and angular) and we are still missing the $\dot{omega}^d$ term. $\dot{omega}^d$ can be obtained in a similar way to the desired angular velocity, by differentiating the previous relationship:

$$\dot{\omega}^d = T_\omega(\Phi^d)\ddot{\Phi}^d + \dot{T_\omega}(\Phi^d, \dot{\Phi}^d)\dot{\Phi}^d \tag{24}$$

Note that the postural task is no longer needed, because we are assigning a $6D$ task. Therefore, the full control law is:

$$F^d = \ddot{p}^d + K_x(p^d - p) + D_x(\dot{p}^d - \dot{p}) \tag{25}$$

$$\Gamma^d = \dot{\omega}^d + K_o e_o + D_o(\omega^d - \omega) \tag{26}$$

$$W^d = \begin{bmatrix} F^d \\ \Gamma^d \end{bmatrix} \tag{27}$$

$$\tau = J^T \left(\Lambda W^d + \mu \right) \tag{28}$$

Verify that the tracking is almost perfect (apart from the transient due to the difference from the initial state and the reference) both during the motion and at steady-state, showing a perfect cancellation of all nonlinear terms, making the system linear and decoupled. An important consequence is that you can now increase the damping gain $D_o = I_{6\times6}30$ beyond 30 without making the system unstable.