Robotics Lab: Homework 3

Implement a vision-based task

Mario Selvaggio

This document contains the homwework 3 of the Robotics Lab class.

Implement a vision-based task

The goal of this homework is to implement a vision-based controller for a 7-degrees-of-freedom robotic manipulator arm into the Gazebo environment. The `ros2_kdl_package` package (link here) and the `ros2_iiwa` package (link here) must be used as starting point. The student is requested to address the following points and provide a detailed report of the employed methods. In addition, a personal github repo with all the developed code must be shared with the instructor. The report is due in one week from the homework release.

1. Construct a gazebo world inserting a blue colored circular object and detect it via the `vision_opencv` package (link here). A template for the implementation is provided here.
   (a) Go into the `iiwa_description` package of the `ros2_iiwa` stack. There you will find a folder `gazebo/models` containing the aruco marker model for gazebo. Taking inspiration from this, create a new model named `spherical_object` that represents a 15 cm radius blue colored spherical object and import it into a new Gazebo world as a static object in $x = 1$, $y = -0.5$, $z = 0.6$. Save the new world into the `/gazebo/worlds/` folder.
   (b) Equip the robot with a camera at the end-effector that loads optionally setting arguments from the `iiwa.launch.py` file (it is recommended to use `<xacro:if value="${use_vision}">`. Modify the launch file to load the robot with the camera into the new world specifying the argument `use_vision:=true`. make sure the robot sees the imported object with the camera, otherwise modify its initial configuration (**Hint:** check it with `rqt_image_view`).
   (c) Once the object is visible in the camera image, use the `ros2_opencv` package (and specifically the `ros2_opencv_node.cpp`) to subscribe to the simulated image, detect the spherical object in it using openCV functions[1], and republish the processed image.
2. Implement a look-at-point vision-based controller
   (a) Spawn the robot with the velocity command interface into a world containing an aruco tag. In the `ros2_kdl_package` package create a `ros2_kdl_vision_control.cpp` node that implements a vision-based controller for the simulated iiwa robot. The controller should be able to perform the following two tasks (it is recommended to switch between the two on the basis of a `task:= positioning|look-at-point` argument passed to the node)
      i. aligns the camera to the aruco marker with a desired position and orientation offsets
      ii. performs a look-at-point task using the following control law

$$\dot{q} = k(LJ_c)^{\dagger} s_d + N\dot{q}_0, \tag{1}$$

where $s_d = [0,\ 0,\ 1]$ is a desired value for

$$s = \frac{{}^cP_o}{||{}^cP_o||} \in \mathbb{S}^2, \tag{2}$$

that is a unit-norm axis connecting the origin of the camera frame and the position of the object ${}^cP_o$. The matrix J_c is the camera Jacobian (to be computed), while $L(s)$ maps linear/angular velocities of the camera to changes in s

$$L(s) = \begin{bmatrix} -\frac{1}{||{}^cP_o||}\left(I - ss^T\right) & S(s) \end{bmatrix} R \in \mathbb{R}^{3\times 6} \quad \text{with} \quad R = \begin{bmatrix} R_c & 0 \\ 0 & R_c \end{bmatrix}, \tag{3}$$

[1] https://learnopencv.com/blob-detection-using-opencv-python-c/

where $S(\cdot)$ is the skew-simmetric operator, R_c the current camera rotation matrix. Finally, $N = \big(I - (LJ)^{\dagger}LJ\big)$ is the matrix spanning the null space of the LJ matrix.

Show the tracking capability by manually moving the aruco marker around via the gazebo user interface and reporting the velocity commands sent to the robot.

(b) Develop a dynamic version of the vision-based contoller. Track the reference velocities generated by the look-at-point vision-based control law with the joint space and the Cartesian space inverse dynamics controllers developed in the previous homework.

Merge the two controllers and enable the joint tracking of a linear position trajectory and the look-at-point vision-based task. **Hint:** Replace the orientation error e_o with respect to a fixed reference (used in the previous homework), with the one generated by the look-at-point vision-based controller. Plot the results in terms of commanded joint torques and Cartesian error norm along the performed trajectory.