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 `kdl_robot` package (at the following link: https://github.com/mrslvg/kdl_robot) 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 instuctor. The report is due in one week from the homewerk release.

1. Construct a gazebo world inserting a circular object and detect it via the `opencv_ros` package
   (a) Go into the `iiwa_gazebo` package of the `iiwa_stack`. There you will find a folder `models` containing the aruco marker model for gazebo. Taking inspiration from this, create a new model named `circular_object` that represents a 15 cm radius colored circular object and import it into a new Gazebo world as a static object at x=1, y=-0.5, z = 0.6 (orient it suitably to accomplish the next point). Save the new world into the `/iiwa_gazebo/worlds/` folder.
   (b) Create a new launch file named `launch/iiwa_gazebo_circular_object.launch` that loads the iiwa robot with `PositionJointInterface` equipped with the camera into the new world via a `launch/iiwa_world_circular_object.launch` file. Make sure the robot sees the imported object with the camera, otherwise modify its configuration (**Hint:** check it with `rqt_image_view`).
   (c) Once the object is visible in the camera image, use the `opencv_ros/` package to detect the circular object using open CV functions. Modify the `opencv_ros_node.cpp` to subscribe to the simulated image, detect the object via openCV functions[1], and republish the processed image.
2. Modify the look-at-point vision-based control example
   (a) The `kdl_robot` package provides a `kdl_robot_vision_control` node that implements a vision-based look-at-point control task with the simulated iiwa robot. It uses the `VelocityJointInterface` enabled by the `iiwa_gazebo_aruco.launch` and the `usb_cam_aruco.launch` launch files. Modify the `kdl_robot_vision_control` node to implement a vision-based task that aligns the camera to the aruco marker with an appropiately chosen position and orientation offsets. Show the tracking capability by moving the aruco marker via the interface and plotting the velocity commands sent to the robot.
   (b) An improved look-at-point algorithm can be devised by noticing that the task is belonging to $\mathbb{S}^2$. Indeed, if we consider

$$s = \frac{{}^cP_o}{||{}^cP_o||} \in \mathbb{S}^2, \tag{1}$$

this is a unit-norm axis. The following matrix maps linear/angular velocities of the camera to changes in s

$$L(s) = \left[-\frac{1}{||{}^cP_o||} \left(I - ss^T \right) \quad S(s) \right] R \in \mathbb{R}^{3\times 6} \quad \text{with} \quad R = \begin{bmatrix} R_c & 0 \\ 0 & R_c \end{bmatrix}, \tag{2}$$

where $S(\cdot)$ is the skew-simmetric operator, R_c the current camera rotation matrix. Implement the following control law

$$\dot{q} = k(LJ)^{\dagger} s_d + N\dot{q}_0, \tag{3}$$

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

where s_d is a desired value for s, e.g. $s_d = [0, 0, 1]$, and $N = \left(I - (LJ)^{\dagger} LJ\right)$ being the matrix spanning the null space of the LJ matrix. Verify that the for a chosen $\dot{q}_0$ the s measure does not change by plotting joint velocities and the s components.

(c) 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. To this end, you have to merge the two controllers and enable the joint tracking of a linear position trajectory and the 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 vision-based controller. Plot the results in terms of commanded joint torques and Cartesian error norm along the performed trajectory.