This document contains the homework 2 of the Robotics Lab class.

Control your robot

The goal of this homework is to develop kinematic and a vision-based controller for a robotic manipulator arm in the simulation environment using `KDL` and `aruco_ros`. 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 problems and provide a detailed point-to-point solution report. A personal github repo containing all the code and a README file detailing how to build and run it must be shared with the instructor (including the URL in the report). The report is due in one week from the homework release.

1. Kinematic control

   (a) Modify the `ros2_kdl_node` such that the following variables become ROS2 params: `traj_duration` , `acc_duration`, `total_time`, `trajectory_len`, `Kp`, and the three components of the trajectory `end_position`. Create a `launch` file that starts the `ros2_kdl_node` loading a `.yaml` file (from a `config` folder) that contains the aforementioned parameters' definition. Add the launch command to the README file in your repo.

   (b) Create a new controller in the `kdl_control` class called `velocity_ctrl_null` that implements the following velocity control law:

   $$\dot{q} = J^{\dagger} K_p e_p + (I - J^{\dagger} J)\dot{q}_0 \tag{1}$$

   where $J^{\dagger}$ is the Jacobian pseudoinverse, e_p is the position error and $\dot{q}_0$ is the joint velocity that keeps the manipulator far from joint limits

   $$\dot{q}_0 = \nabla \sum_{i=1}^{n} \frac{1}{\lambda} \frac{\left(q_i^+ - q_i^-\right)^2}{\left(q_i^+ - q_i(t)\right)\left(q_i(t) - q_i^-\right)}, \tag{2}$$

   where λ is a scaling factor, q_i^+ and q_i^- are the $i-$th upper and lower joint limit, respectively. Test this new control mode (with velocity the corresponding cmd interface) and compare to the previous velocity control by reporting the plots of the commanded velocities and the joint position values. Switch between the two velocity controllers creating an additional parameter `ctrl:=velocity_ctrl|velocity_ctrl_null` passed as argument to the node. Insert this in the previous launch file.

   (c) Following this tutorial, make our `ros2_kdl_node` an action server that executes the same linear trajectory and publishes the position error as feedback. Write an action client to test your code and add the command to run both of them to the README file in your repo.

2. Vision-based control

   (a) Construct a gazebo world inserting an aruco tag and detect it via the `aruco_ros` package (link here). Go into the `iiwa_description` package of the `ros2_iiwa` stack. There, create a folder `gazebo/models` containing the aruco marker model for gazebo. Create a new model named `aruco_tag` and import it into a new Gazebo world as a static object in a position that is visible by the camera. Save the new world into the `/gazebo/worlds/` folder.

   (b) Spawn the robot with the velocity command interface into the world containing the aruco tag. In the `kdl_control` class of the `ros2_kdl_package` package create a vision-based controller called `vision_ctrl` for the simulated iiwa robot that is activated setting the `ctrl` ROS parameter to

`vision`. Create a subscriber to the aruco marker pose published by the `aruco_ros` package[1]. The controller should be able to perform a look-at-point task using the following control law

$$\dot{q} = K(L(s)J_c)^{\dagger} s_d + N\dot{q}_0, \tag{3}$$

where K is a diagonal gain matrix, $s_d = [0,\, 0,\, 1]$ is a desired value for

$$s = \frac{{}^{c}P_o}{||{}^{c}P_o||} \in \mathbb{S}^2, \tag{4}$$

that is a unit-norm axis connecting the origin of the camera frame and the position of the object ${}^{c}P_o$. The matrix J_c is the camera Jacobian (different from the end-effector one), while $L(s)$ maps linear/angular velocities of the camera to changes in s as follows

$$L(s) = \left[-\frac{1}{||{}^{c}P_o||} \left(I - ss^{\mathrm{T}}\right) \quad S(s) \right] R \in \mathbb{R}^{3\times 6} \quad \text{with} \quad R = \begin{bmatrix} R_c^{\mathrm{T}} & 0 \\ 0 & R_c^{\mathrm{T}} \end{bmatrix}, \tag{5}$$

where $S(\cdot)$ is the skew-symmetric operator, R_c the current camera rotation matrix. Finally, $N = \left(I - (L(s)J_c)^{\dagger} L(s)J_c\right)$ is the matrix spanning the null space of the $L(s)J_c$ matrix.

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

(c) Create a ROS 2 service to update the aruco marker position in Gazebo. To do so, starting from the `/set_pose` ign service, create a `parameter_bridge` in the previously created launch file. Test the bridged service via ROS 2 service call.

[1] **Note:** the aruco pose is given with respect to the image frame, not the frame attached to your camera link!