HiWI Position Programming Robots: Software Engineer Support

April 11, 2023

The goal of this assignment is to test your understanding of some fundamental concepts in programming, source control, and the ability to navigate concepts pertaining to robotic manipulators. Please keep track of approximately how much you spend on this and note it when you submit your solution.

1 Instructions

- You will have **10 days** to complete this test.
- Your submission should contain a local git repository and a remote. The changes you make to the project should be tracked and committed in a organized fashion in order to demonstrate your workflow.
- If certain assumptions need to be made for a specific problem, please go ahead and report them later in a separate file.

2 Required Software(s) & Problems

The following software and libraries are needed and should be installed in the system before starting:

- Ubuntu 20.04 LTS
- MATLAB (preferably R2021b or newer)
- DQ Robotics library[1]
- CoppeliaSim[2]
- MuJoCo[3]

[1]Installation instructions: https://dqrobotics.github.io/
[2]Installation instructions: https://www.coppeliarobotics.com/
[3]Installation instructions: https://mujoco.org/

- ROS Noetic[4]

To test out the environment, and warm-up a bit consider the provided example files:

1. Open CoppeliaSim and load the *franka_ kinematic.ttt* scene.
2. Run *main.m* and you should see the robot moving from the initial configuration to the goal.

Now that you have properly configured the required software, your tasks are:

1. Modify the provided MATLAB code such that the user can change the parameters to make the robot converge faster or slower;
2. Given the desired end-effector pose $\underline{\boldsymbol{x}}_d$ and the error defined as $\underline{\boldsymbol{e}} = \underline{\boldsymbol{x}} - \underline{\boldsymbol{x}}_d$, create a new MATLAB code where the controller is replaced by the control law,

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger} k \operatorname{vec}_8 \underline{\boldsymbol{e}} + \boldsymbol{N} d \left(\boldsymbol{q}_c - \boldsymbol{q} \right), \tag{1}$$

   where $\boldsymbol{J}^{\dagger}$ is the Moore-Penrose pseudoinverse of the Jacobian matrix $\boldsymbol{J}$, the proportional gains are $k, d > 0$, the nullspace projector is $\boldsymbol{N} = \boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J}$, in which $\boldsymbol{I}$ is the identity matrix, and the joint center $\boldsymbol{q}_c = 0.5 \left(\boldsymbol{q}_{\min} + \boldsymbol{q}_{\max} \right)$, with $\boldsymbol{q}_{\min}$ and $\boldsymbol{q}_{\max}$ designating, respectively, the minimum and maximum positions of the robot joints.
3. Convert the code of the original simulation to C++/ROS;
4. Run this C++/ROS code and save the data for plotting in MATLAB;
5. Convert the code of the simulation using (1) to C++/ROS;
6. Run the simulation with any of the controllers in MuJoCo; Record a screen video and save it in .mp4/other format.
7. Feasibility checks, with regards to viable inverse kinematic (IK) solutions are common in manipulation planning tasks. Pick two among the following most commonly used IK solvers:
   (a) IKFast[5]
   (b) Pinocchio[6]
   (c) Analytical IK solver for Franka Emika Panda[7]

[4] Installation instructions: https://www.ros.org/

[5] https://github.com/ros-planning/moveit_tutorials/blob/kinetic-devel/doc/ikfast/ikfast_tutorial.rst

[6] https://gepettoweb.laas.fr/doc/stack-of-tasks/pinocchio/master/doxygen-html/md_doc_b-examples_i-inverse-kinematics.html

[7] https://github.com/ffall007/franka_analytical_ik

(a) Implement a C++ code that allows the user to receive the IK solution for the Franka Emika Panda given a specified end-effector pose using either one of the selected two. (b) From the selection, report the (computation) timings and other performance metrics if possible.

8. (**Bonus**) Get the dynamic parameters using Pinocchio and report the parameters like Mass matrix (for a specific configuration $\boldsymbol{q}$), Coriolis matrix, and gravity vector in a file.

9. (**Bonus**) Using the dynamic parameters obtained in the last step you will now implement a *computed torque control law*. To this end, if our desired trajectory $\boldsymbol{q}_d(t)$ is selected for the arm manipulation, the tracking error $\boldsymbol{e}(t)$ can be defined as,

$$\boldsymbol{e}(t) = \boldsymbol{q}_d(t) - \boldsymbol{q}(t), \tag{2}$$

The final control law is then,

$$\boldsymbol{\tau} = M(\boldsymbol{q})(\ddot{\boldsymbol{q}}_d + K_d\dot{\boldsymbol{e}} + K_p\boldsymbol{e}) + C(\boldsymbol{q}, \dot{\boldsymbol{q}}) + g(\boldsymbol{q}). \tag{3}$$

The gains K_p and K_d are usually selected for critical damping. For a more in-depth overview, please take a look at these informative slides.[8].

10. (**Bonus**) Finally, resolve redundancy at the kinematic level using a Quadratic program (QP). Please have a look at this useful blog[9] for some references/equations.

3 Submit your solutions

Zip or tar your project directory including all generated files and video(s). Attach your archive along with any comments/issues within **10 days**.

[8] Advanced Robotics - Computed Torque Control
[9] Stephane Caron's Blog-post introducing QP's