RBE 502 — Robot Control

Instructor: Siavash Farzan

Fall 2022

Programming Assignment 4:

Robust Control of the RRBot Robotic Arm

4.1 Overview

The RRBot robot – for which we have studied its dynamics, trajectory generation and controls in Programming Assignments 1, 2 and 3 – is a simple two-link robot arm with two revolute joints.

You should have already installed and built the RRBot ROS package in Programming Assignment 0. To visualize the robot in Gazebo, you can run the following command in the Ubuntu terminal:

```
roslaunch rrbot_gazebo rrbot_world.launch
```

In Programming Assignment 3, we generated a dynamically feasible trajectory for the system and designed a feedback linearization-based control to track the trajectory and drive the robot in Gazebo to the upward equilibrium point. In this assignment, we design a *robust* control algorithm to track the same trajectory in the presence of *dynamic model uncertainties*.

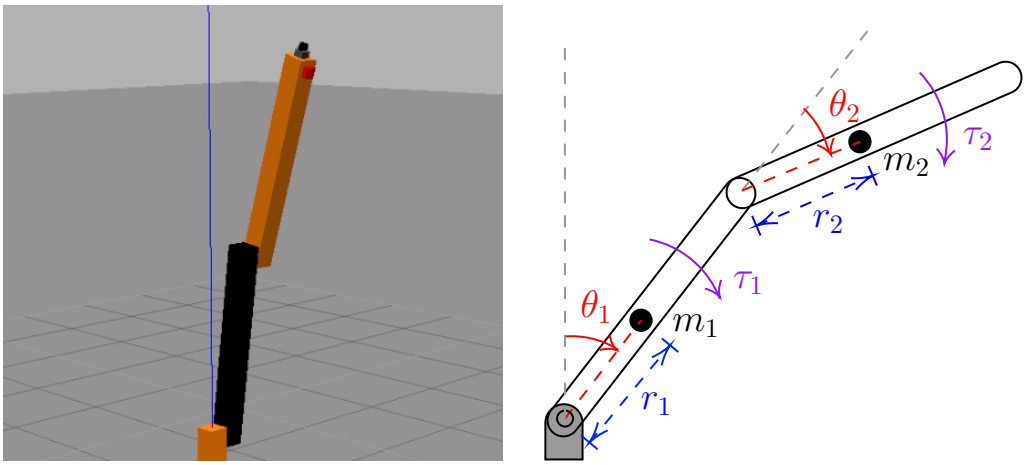

Figure 1: 2-DoF Revolute-Revolute robot arm (RRBot)

4.2 Problem Statement

Consider the 2-DoF Revolute-Revolute robot arm (RRBot) shown in Figure 1. Each joint is equipped with an actuator, applying control torques τ_1 and τ_2 to joint 1 and joint 2, respectively. Lengths of the first and second link are l_1 and l_2, respectively. The links have distributed

mass, and r_1 and r_2 denote the location of the center of mass for each link (with mass of m_1 and m_2). The moments of inertia of link 1 and link 2 are I_1 and I_2. θ_1 is the angle of the first link with respect to the vertical axis, and θ_2 is the angle of the second link with respect to the first link.

The *actual* physical parameters of the robot are listed below:

$$m_1 = m_2 = 1\ (kg), \quad l_1 = l_2 = 1\ (m), \quad r_1 = r_2 = 0.45\ (m)$$
$$I_1 = I_2 = 0.084\ (kg \cdot m^2), \quad g = 9.81\ (m/s^2)$$

Important note: For all the tasks to follow, the joint torques must stay within the bounds of:

$$-20 \leq u_1 \leq 20\ (Nm), \qquad -10 \leq u_2 \leq 10\ (Nm)$$

Violation of the above actuator limits will be considered as a control failure.

4.2.1 Initial Setup

a) Use your code from Programming Assignment 3 to generate a cubic polynomial trajectory for the first and second joints of the robot. The time span and the desired initial and final joint angles and velocities are given by:

$$t_0 = 0, \quad t_f = 10\ sec$$
$$\theta_1(t_0) = 180^\circ, \quad \theta_1(t_f) = 0, \quad \theta_2(t_0) = 90^\circ, \quad \theta_2(t_f) = 0$$
$$\dot{\theta}_1(t_0) = \dot{\theta}_1(t_f) = \dot{\theta}_2(t_0) = \dot{\theta}_2(t_f) = 0$$

Note: this part is already done in Programming Assignment 3.

b) Consider the equations of motion derived for the robot in Programming Assignment 1. As discussed in Programming Assignment 3, the equations of motion can be transformed into the standard Manipulator form:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau$$

For the purpose of this assignment, the robot's equations of motion in the manipulator from are provided in the Appendix. Copy and paste the provided equations of motion into a MATLAB script (with appropriate symbols defined) to obtain symbolic expressions for the robot's dynamics.

4.2.2 Robust Control

For the robust control design in this assignment, we assume that the mass and inertia parameters of the robot are unknown, but the nominal values are given by:

$$\hat{m}_1 = \hat{m}_2 = 0.75\ (kg), \qquad \hat{I}_1 = \hat{I}_2 = 0.063\ (kg \cdot m^2)$$

c) ***(7 points)*** Design a Robust Inverse Dynamics control law for trajectory tracking by the robot using the method described in Lecture 19. The control gains to be designed are $K_p \in \mathbb{R}^{2\times2}$ and $K_d \in \mathbb{R}^{2\times2}$ (for the virtual control input v), and the Lyapunov matrix $P = P^T > 0 \in \mathbb{R}^{4\times4}$ for the robust control term v_r.

- Use state-feedback control design to determine the control gains K_p and K_d to place the eigenvalues at $\{-3, -3, -4, -4\}$. You can use the `place` function in MATLAB to design the control gain matrix $K \in \mathbb{R}^{2\times 4}$, where $K = \begin{bmatrix} K_p & K_d \end{bmatrix}$.
- The matrix P is the solution to the Lyapunov equation $A^T P + PA = -Q$. You can use the `lyap` function in MATLAB to solve for P.

Moreover, initialize a constant value for the uncertainty upper bound ρ, which is used to compute the robust control term v_r. This bound can be later tuned in simulation by trial and error. For the purpose of this assignment, you can use a *constant* ρ value.

d) ***(2 points)*** Update the `ode` function developed in Programming Assignment 3 to implement the robust inverse dynamics control law designed in part (c). Note that you will need to evaluate the cubic polynomial trajectories inside the `ode` function to obtain the desired states at each point in time.
Note: You would need to implement an `ode` function to generate the response of the system given the overall robust control input τ. In this `ode` function, you should use the *true values* for physical parameters when integrating the equations of motion. However, in the control law, you should NOT use the true parameters values, but *only the nominal values* given above.

e) ***(3 points)*** Use `ode45` and the `ode` function developed in part (d) to construct a simulation of the system in MATLAB with the time span of $[0, 10]$ sec and initial conditions of:

$$\theta_1(0) = 200^\circ, \quad \theta_2(0) = 125^\circ, \quad \dot{\theta}_1 = 0, \quad \dot{\theta}_2 = 0$$

Evaluate the performance of the controller for tracking the desired trajectories generated in Section 4.2.1. Plot the state trajectories, the control inputs trajectories, and the associated desired trajectories to evaluate the performance.
If the performance is not satisfactory (i.e. the system does not track and converge to the desired trajectory), go back to part (c) and tune the design parameters, including the uncertainty upper bound ρ, the state-feedback control gains, or the Lyapunov matrix P.
Hint: The control inputs can be reconstructed after the solution is returned by `ode45`.

f) ***(3 points)*** Include a boundary layer in the robust control term v_r to reduce the chattering effect observed in the control input in part (e). Perform the same simulation as part (e), and plot the state trajectories, the control inputs trajectories, and the associated desired trajectories to evaluate the performance. Discuss your findings in your final report.

g) ***(1 points)*** To evaluate the performance of the robot without the *robust* inverse dynamics control, construct a simulation of the system in MATLAB with the same control law but with the v_r term set to zero (do not change other design parameters such as K_p and K_d). Again, plot the state trajectories, the control inputs trajectories and the associated desired trajectories to compare the resulting performance with the performance obtained in part (e). Discuss the results in your final report.

h) ***(4 points)*** Create a new copy of the `rrbot_control.m` file provided in Programming Assignment 2, and rename the new file to `rrbot_robust_control.m`.
Note: Alternatively, you can use the ROS Python script posted on Canvas for this purpose.
Update the code inside the `while` loop to control the RRbot robot in Gazebo for 10 seconds using your robust inverse dynamics controller with boundary layer designed in part (f). Sample the data (joint angles, joint velocities and control inputs) at each loop to be plotted at

the end. Feel free to define new functions and variables in your program if needed. If the Gazebo performance is not satisfactory, re-tune the design parameters to achieve the desired performance. Compare the resulting trajectories and control inputs in Gazebo with those obtained in part (f) in MATLAB. Discuss your findings in your final report.

4.3 Robot Controller Setup in Gazebo

Before testing your control design on the RRbot robot in Gazebo, you need to setup your ROS controller nodes for the robot by following the parts in Section 2.3 of Programming Assignment 2.

We are now ready to evaluate the controller performance in Gazebo.

4.4 Performance Testing in Gazebo

i) Open a terminal in Ubuntu, launch the Gazebo simulator and spawn a new robot:

```
roslaunch rrbot_gazebo rrbot_world.launch
```

ii) Once the robot is successfully spawned in Gazebo, in a new terminal, launch the rrbot control node:

```
roslaunch rrbot_control rrbot_effort_control.launch
```

iii) We can now evaluate the performance of the robust control algorithm by running the `rrbot_robust_control.m` script in MATLAB.

4.5 Submission

- Submission: individual submission via Gradescope. Submit the MATLAB (`.m`) files you have written as a zip folder and a detailed report including:
  - mathematical formulation, derivation, and expressions of the control laws formulated in part (c);
  - a list of all the design (tuning) parameters and their associated values used in the simulations, along with a discussion on the effect of each parameter on the control performance;
  - the plots generated in parts (e), (f), (g), (h), with sufficient discussion and comparison.
- Due: as specified on Canvas.
- Files to submit: (please use the exact file name, and do NOT include the PDF file in the zip folder)
  - `LastName_ProgAssignment4_report.pdf`
  - `LastName_ProgAssignment4_matlab.zip`

4.6 Appendix

Consider the equations of motion in the manipulator form

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau.$$

The dynamics of the robot with the generalized coordinates $q = \begin{bmatrix} q_1 & q_2 \end{bmatrix}^T := \begin{bmatrix} \theta_1 & \theta_2 \end{bmatrix}^T$ and the generalized forces $\tau = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^T$ can be written as:

```
Mmat= [a+2*b*cos(q2), d+b*cos(q2);  d+b*cos(q2), d];
Cmat= [-b*sin(q2)*dq2, -b*sin(q2)*(dq1+dq2); b*sin(q2)*dq1,0];
Gmat= [-m1*g*r1*sin(q1)-m2*g*(l1*sin(q1)+r2*sin(q1+q2)); -m2*g*r2*sin(q1+q2)];

% where
a = I1 + I2 + m1*r1^2 + m2*(l1^2 + r2^2);
b = m2*l1*r2;
d = I2 + m2*r2^2;
```