Robot Dynamics Midterm 2

Prof. Marco Hutter
Teaching Assistants: Joonho Lee, Ruben Grandia,
Koen Krämer, Takahiro Miki

November 10, 2021

Duration: 1h 15min
Permitted Aids: Everything; no communication among students during the test

1 Instructions

1. Download the ZIP file for midterm 2. Extract all contents of this file into a new folder and set MATLAB's[1] current path to this folder.
2. Run `init_workspace` in the Matlab command line
3. All problem files that you need to complete are located in the `problems` folder
4. Run provided simulations (files named as `simulate_(...).m`) to check your controllers. Set `use_solution` to 1 to see how the solution behaves.
5. Run `run_problems.m` to check if your functions run. This script does not test for correctness. You will get 0 points if a function does not run (e.g., for syntax errors).
6. You can use helper functions provided in `utils` folder. However, do not add to or modify the files in this folder. All your own implementations should go directly into the question files in the `problems` folder. Implementations outside the provided templates will not be graded and receive 0 points.
7. When the time is up, zip the entire folder and name it
   `ETHStudentID_StudentName.zip`
   Submit this zip-file through Moodle under **Midterm Exam 2 Submission**. You should receive a confirmation email.
8. If the previous step did not succeed, you can email your file to
   `robotdynamics@leggedrobotics.com`
   from your ETH email address with the subject line
   `[RobotDynamics] ETHStudentID - StudentName`

[1] Online version of MATLAB at `https://matlab.mathworks.com/`

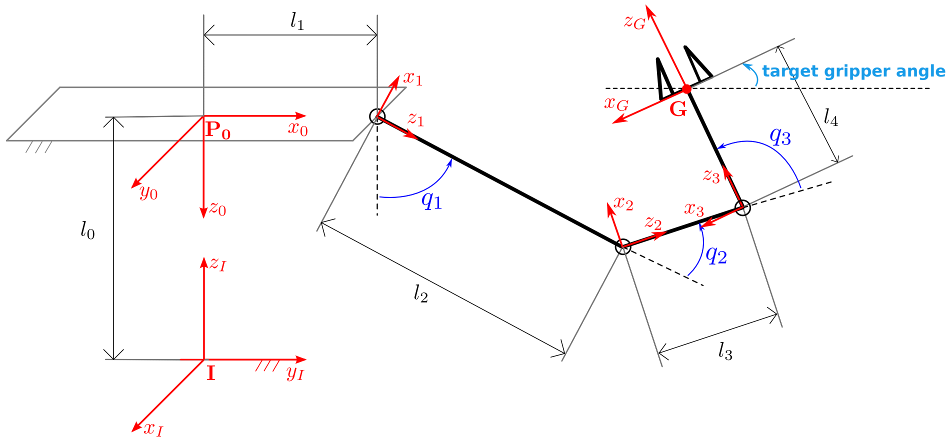

Figure 1: Schematic of a 3 degrees of freedom robotic arm attached to a fixed base. All joints rotate around the positive y_0 axis. The y axis of the frames $\{1\}, \{2\}, \{3\}$ is parallel to the y_0 axis.

2 Questions

You will model the dynamics of the robot arm shown in Fig. 1 and use it for control. It is a 3 degrees of freedom arm connected to a **fixed** base.

Let $\{0\}$ be the base frame, which is displaced by l_0 from the inertial frame $\{I\}$ along the ${}_Iz$ axis. The arm is composed of three links. The reference frames attached to each link are denoted as $\{1\}, \{2\}, \{3\}$. The links' segments have lengths l_2, l_3, l_4. The generalized coordinates are defined as

$$\boldsymbol{q} = \begin{bmatrix} q_1 & q_2 & q_3 \end{bmatrix}^\top . \tag{1}$$

In the following questions, we have already provided the kinematics (transforms, Jacobians) and controller gains (k_P, k_D) for you. The variables stored as Matlab *cells* may be accessed as follows:

```
m{k};           % mass of link k
k_r_ks{k};      % Position of com of link k in frame k
k_I_s{k};       % Rotational inertia of link k in frame k
R_Ik{k};        % Rotation matrix from frame k to I
I_Jp{k};        % Positional Jacobian of link k in I frame
I_Jr{k};        % Rotational Jacobian of link k in I frame
etc..
```

Question 1. 3 P.

Calculate the mass matrix $\boldsymbol{M}(\boldsymbol{q})$, nonlinear terms $\boldsymbol{b}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ (Coriolis and centrifugal), and gravitational terms $\boldsymbol{g}(\boldsymbol{q})$. A helper function, `dAdt.m`, is available in the `utils` folder, to compute the time derivative of a matrix.

Implement your solution in `Q1_generate_eom.m`.

Question 2. 2 P.

Implement a forward dynamics simulator that computes the joint accelerations $\ddot{\boldsymbol{q}}$ and integrates them to get $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. You should implement the calculation of $\ddot{\boldsymbol{q}}$, given $\boldsymbol{\tau}$, the input torque for each joint.

Use the mass matrix, non-linear terms and gravitational terms obtained from `M_fun_solution(q)`, `b_fun_solution(q, dq)` and `g_fun_solution(q)`.

You should implement your solution in `Q2_forward_dynamics.m`.

Question 3. 2 P.

Implement a joint-level PD controller that compensates for the gravitational terms and tracks desired joint positions and velocities. Calculate $\boldsymbol{\tau}$, the control torque for each joint.

Current joint positions $\boldsymbol{q}$ and joint velocities $\dot{\boldsymbol{q}}$, as well as desired joint positions $\boldsymbol{q}^d$ and desired joint velocities $\dot{\boldsymbol{q}}^d$ are given to the controller as input arguments to the Matlab file. You should obtain the gravitational terms in this question through the provided `g_fun_solution(q)`. Use the provided PD gains.

Implement your solution in `Q3_gravity_compensation.m`. A simulation of your implemented controller (or the solution) is available in `simulate_gravity_compensation.m`. Set `use_solution` to 1 to see how the solution behaves.

Question 4. 3 P.

Implement a controller that uses a task-space inverse dynamics algorithm that makes use of the end-effector dynamics. In this question you have to implement **(1) end-effector dynamics** and a **(2) end-effector motion controller**.

The inputs to this controller are the desired motion of the point G of the gripper as well as the current joint position $\boldsymbol{q}$ and joint velocities $\dot{\boldsymbol{q}}$. The desired motion is specified by the following components (all expressed in the inertial frame):

- ${}_I r_{IG}^d \in \mathbb{R}^3$: desired gripper position
- ${}_I v_G^d \in \mathbb{R}^3$: desired gripper velocity
- ${}_I a_G^d \in \mathbb{R}^3$: desired gripper acceleration
- $\mathrm{C}_{IG}^d \in SO(3)$: desired gripper orientation, specified as a 3×3 rotation matrix.

Implement a controller that computes the torques necessary for following the desired linear acceleration of the end-effector in task-space as well as a feedback on the orientation, position, and velocity of the gripper. The desired motion is passed as input arguments to the Matlab file. The PD gains are provided. Use the mass matrix, non-linear terms and gravitational terms obtained from `M_fun_solution(q)`, `b_fun_solution(q, dq)` and `g_fun_solution(q)`.

Implement your solution in `Q4_task_space_control.m`. A simulation of your implemented controller (or the solution) is available in `simulate_task_space_control.m`. Set `use_solution` to 1 to see how the solution behaves.

Note: Use the provided `pseudoInverseMat()` function when implementing the projected inertia (Λ) for numerical stability.