Robot Dynamics Quiz 1

Prof. Marco Hutter
Teaching Assistants: Jan Preisig, Ruben Grandia,
Jean Pierre Sleiman, Maria Vittoria Minniti

October 14, 2020

Duration: 1h 15min
Permitted Aids: The exam is open book, which means you can use the script, slides, exercises, etc; The use of internet (beside for licenses) is forbidden; no communication among students during the test.

1 Instructions

1. Download the ZIP file for quiz 1 from Piazza. 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 `evaluate_problems` 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).
5. When the time is up, zip the entire folder and name it
   `ETHStudentID_StudentName.zip`
   Upload this zip-file through the following link
   `https://www.dropbox.com/request/JGp7ImPmEZzRDdrssfyy`.
   You will receive a confirmation of receipt.
6. 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/`

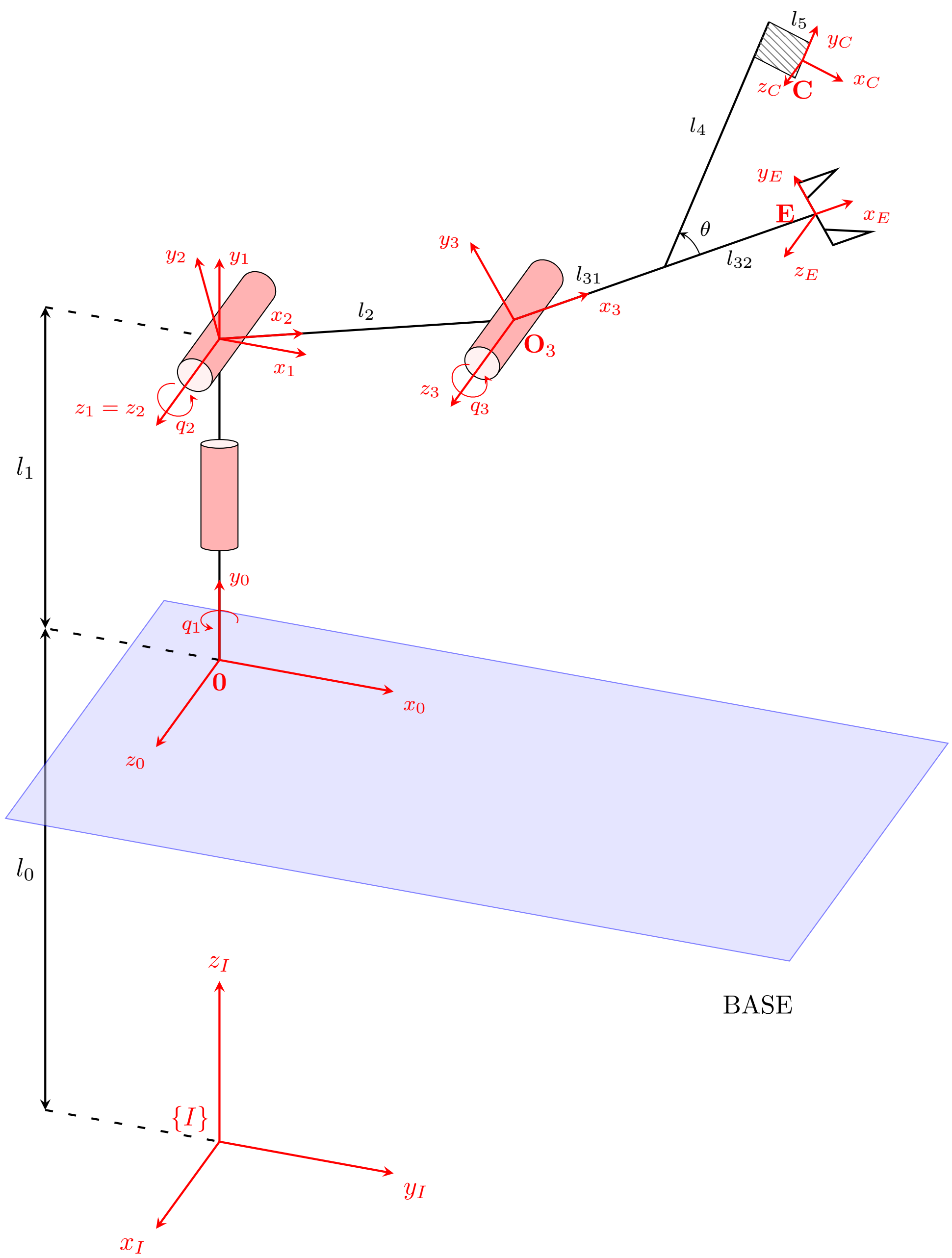

Figure 1: Schematic of a 3 degrees of freedom robotic arm attached to a fixed base. A camera is rigidly mounted on the last link of the arm.

2 Questions

In this quiz, you will model the forward, differential, and inverse kinematics of the robotic arm shown in Fig. 1. 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 ${}_I z$ 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_1, l_2, $l_{31} + l_{32}$.
Additionally, a camera is mounted on the last link of the arm. As shown in figure 1, the camera link is mounted at a constant angle θ around the axis z_3.

A visualization of the robot in the plane $x_1 y_1$ is also provided in figure 2.

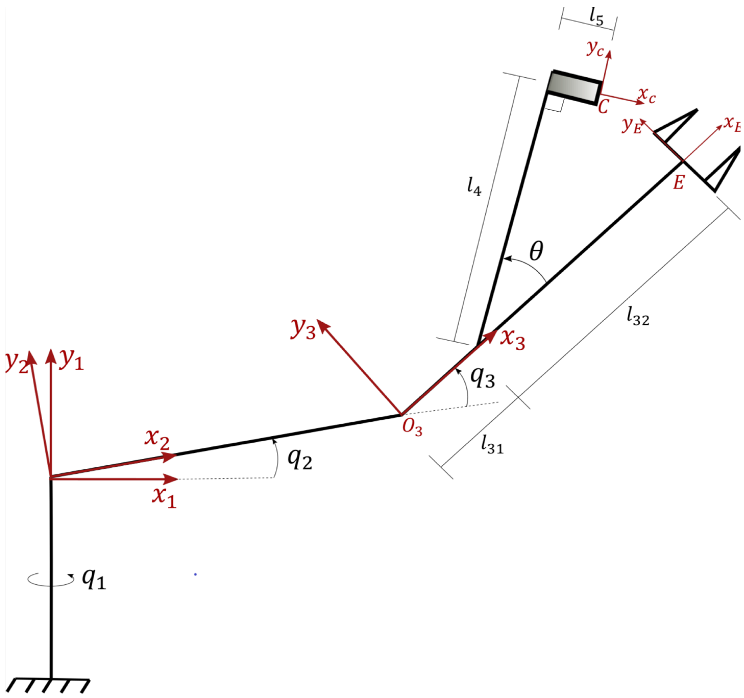

Figure 2: Planar visualization of the 3-DOF robotic arm

The generalized coordinates are defined as

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

NOTE: the joint angles q_2 and q_3 are assumed to be zero when the axis x_2 and x_3 are parallel to the axis x_0 of the base frame.
In the following questions, all required parameters are passed to your functions in a structure called **params**. You can access it as follows:

```
l0 = params.l0;
l1 = params.l1;
l2 = params.l2;
l31 = params.l31;
l32 = params.l32;
l4 = params.l4;
l5 = params.l5;
theta = params.theta;
max_it = params.max_it;
lambda = params.lambda;
alpha = params.alpha;
```

Question 1. 6 P.

Let $\{E\}$ be the end-effector frame. Find the homogeneous transform between the inertial frame $\{I\}$ and the end-effector frame $\{E\}$, i.e., the matrix $\mathbf{T}_{IE}$ as a function of the generalized coordinates $\boldsymbol{q}$.
Hint: Try to find the transforms of subsequent frames first.

You should implement your solution in the function `jointToEndeffectorPose.m`

Solution 1.

```
function [ T_IE ] = jointToEndeffectorPose( q, params )
  % q: a 3x1 vector of generalized coordinates
  % params: a struct of parameters

  % Link lengths (meters)
  l0 = params.l0;
  l1 = params.l1;
  l2 = params.l2;
  l31 = params.l31;
  l32 = params.l32;

  % Joint positions
  q1 = q(1);
  q2 = q(2);
  q3 = q(3);

  % compute T_I0
  p_I0_I =  [0; 0; l0];
  C_I0 = [0 0 1;
          1 0 0;
          0 1 0];
  T_I0 = [C_I0 p_I0_I;
          0 0 0 1];

  % compute T_01
  p_01_0 = [0; l1; 0];
  C_01 = [cos(q1) 0 sin(q1);
          0 1 0;
          -sin(q1) 0 cos(q1)];
  T_01 = [C_01 p_01_0;
          0 0 0 1];
  % compute T_12
  p_12_1 = [0; 0; 0];
  C_12 = [cos(q2) -sin(q2) 0;
          sin(q2) cos(q2) 0;
          0 0 1];
  T_12 = [C_12 p_12_1;
          0 0 0 1];
  % compute T_23
  p_23_2 = [l2; 0; 0];
  C_23 = [cos(q3) -sin(q3) 0;
          sin(q3) cos(q3) 0;
          0 0 1];
  T_23 = [C_23 p_23_2;
          0 0 0 1];
  % compute T_3E
  C_3E = eye(3,3);
  p_3E_3 = [l31 + l32; 0; 0];

  T_3E = [C_3E p_3E_3;
          0 0 0 1];
  % concatenate transformations
  T_IE = T_I0 * T_01 * T_12 * T_23 * T_3E;

```

```
end
```

Question 2. 6 P.

Consider the difference between the geometric Jacobian $\mathbf{J}_{IC} \in \mathbb{R}^{6\times3}$ of point $\mathbf{C}$, and the geometric Jacobian $\mathbf{J}_{IO_3} \in \mathbb{R}^{6\times3}$ of point $\mathbf{O_3}$:

$$ {}_3\mathbf{J}_{O_3C} = {}_3\mathbf{J}_{IC} - {}_3\mathbf{J}_{IO_3}, \tag{2}$$

where all the terms are expressed in the same frame $\{3\}$. The Jacobian ${}_3\mathbf{J}_{O_3C} \in \mathbb{R}^{6\times3}$ defines the following mapping:

$$\begin{bmatrix} {}_3\boldsymbol{v}_{IC} - {}_3\boldsymbol{v}_{IO_3} \\ {}_3\boldsymbol{\omega}_{IC} - {}_3\boldsymbol{\omega}_{I3} \end{bmatrix} = {}_3\mathbf{J}_{O_3C}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{3}$$

where ${}_3\boldsymbol{v}_{IC}, {}_3\boldsymbol{v}_{IO_3}, {}_3\boldsymbol{\omega}_{IC}, {}_3\boldsymbol{\omega}_{I3} \in \mathbb{R}^3$ are the linear and angular velocities of the frames $\{C\}$ and $\{3\}$, respectively. Compute the Jacobian ${}_3\mathbf{J}_{O_3C}$ in reference system $\{3\}$, using the following hints:

1. Note that

   $$\boldsymbol{v}_{IC} - \boldsymbol{v}_{IO_3} = \begin{bmatrix}\mathbf{n}_1 \times \mathbf{r}_{O_3C} & \mathbf{n}_2 \times \mathbf{r}_{O_3C} & \mathbf{n}_3 \times \mathbf{r}_{O_3C}\end{bmatrix}\dot{\mathbf{q}} \tag{4}$$

   where $\mathbf{n_1}, \mathbf{n_2}, \mathbf{n_3} \in \mathbb{R}^3$ are the rotational axes of the three joints, and $\mathbf{r}_{O_3C} \in \mathbb{R}^3$ is the position vector from $\mathbf{O_3}$ to $\mathbf{C}$.

2. The points $\mathbf{C}$ and $\mathbf{O_3}$ belong to the same rigid body.

3. The MATLAB function for cross product $\boldsymbol{a} \times \boldsymbol{b}$ is `cross(a,b)`.

You should implement your solution in the function `point3ToCameraGeometricJacobian.m`

Solution 2.

```
function J_3C_3 = point3ToCameraGeometricJacobian(q, params)
  % Inputs:
  %     q: a 3x1 vector of generalized coordinates
  %     params: a struct of parameters
  % Output:
  %     J_3C_3: geometric Jacobian from point O_3 to point C, ...
      expressed in
  %     frame {3}

  theta = params.theta;
  l31 = params.l31;
  l4 = params.l4;
  l5 = params.l5;

  % Implement your solution here...
  % Joint positions
  q1 = q(1);
  q2 = q(2);
  q3 = q(3);

  % compute rotation matrix from camera frame to frame 3
  C_3C = [cos(theta-pi/2), -sin(theta-pi/2), 0;
          sin(theta-pi/2), cos(theta-pi/2), 0;
          0, 0, 1];
  % position vector from 3 to C in frame 3
  p_3C_3 = [l31; 0; 0] + C_3C*[l5; l4; 0];

  % compute rotation matrix from 3 to 1
```

```
  C_13 = [cos(q2+q3), -sin(q2+q3), 0;
          sin(q2+q3), cos(q2+q3), 0;
          0, 0, 1];

  % compute rotation axes
  n1_3 = C_13'*[0; 1; 0];
  n2_3 = [0; 0; 1];
  n3_3 = [0; 0; 1];

  % write geometric jacobian
  J_3C_3 = zeros(6,3);
  J_3C_3(1:3,:) = [cross(n1_3, p_3C_3), cross(n2_3, p_3C_3), ...
      cross(n3_3, p_3C_3)];
  J_3C_3(4:6,:) = zeros(3,3);

end
```

Question 3. 3 P.

Given a camera position $\mathbf{r}_{IC}^{*} \in \mathbb{R}^3$ and a starting joint configuration $\mathbf{q_0}$, implement a MATLAB function which computes the joint angles $\mathbf{q}$ corresponding to the given camera position, using an iterative inverse kinematics algorithm.
For this question, we provide:

- the position vector ${}_I\mathbf{p}_{IC}$.

  You can access it with `jointToCameraPosition_solution(q, params)`.

- the position Jacobian ${}_I\mathbf{J}_{IC} \in \mathbb{R}^{3\times3}$ of point $\mathbf{C}$.

  You can access it with `jointToPositionJacobian_solution(q, params)`.

- a function for calculating damped pseudo-inverses, as you have seen in the exercise: `pseudoInverseMat_solution(J, lambda)`.

You should implement your solution in the function `inverseKinematics.m`

Solution 3.

```
function [ q ] = inverseKinematics(I_r_IC_des, q_0, tol, params)
% Input:
% I_r_IC_des: 3x1 desired position of the point C
% q_0: 3x1 initial guess for joint angles
% tol: 1x1 tolerance to use as termination criterion
%      The tolerance should be used as:
%      norm(I_r_IC_des - I_r_IC) < tol
% params: a struct of parameters
% Output:
% q: a vector of joint angles q (3x1) which achieves the desired
%    task-space position

% 0. Setup
max_it = params.max_it;        % Set the maximum number of iterations.
lambda = params.lambda;        % Damping factor
alpha = params.alpha;          % Update rate

% 1. start configuration
q = q_0;

% implement your solution here ...
it = 0;
% 2. iterate until terminating condition
while (it==0 || (norm(dr)>tol && it < max_it))
    % 3. evaluate Jacobian for current q
```

```
    I_J = jointToPositionJacobian_solution(q, params);

    % 4. Update the psuedo-inverse
    I_J_pinv = pseudoInverseMat_solution(I_J, lambda);

    % 5. Find the camera configuration error vector
    % position error
    I_r_IC = jointToCameraPosition_solution(q, params);
    dr = I_r_IC_des - I_r_IC;

    % 6. Update the generalized coordinates
    q =  q + alpha*I_J_pinv*dr;

    it = it+1;
end


end
```

Question 4. 3 P.

Assume now that the base frame $\{0\}$ can freely rotate and its rotation with respect to the inertial frame $\{I\}$ is described by a given quaternion $\mathbf{Q}_{I0}$. Write a MATLAB function to compute the rotation matrix $\mathbf{C}_{IC}$, that represents the orientation of the camera frame $\{C\}$ with respect to the inertial frame $\{I\}$.

For this question, we provide the transform $\mathbf{T}_{0C}$, which you can access with `T_0C_solution(q, params)`.

You should implement your solution in the function `cameraFrameOrientationWithBaseRotation.m`

Solution 4.

```
function [ C_IC ] = cameraFrameOrientationWithBaseRotation(qW, qX, ...
    qY, qZ, q, params)
% Input:
%   qW, qX, qY, qZ: components of the quaternion q_I0 = [qW, qX, ...
    qY, qZ],
%                   which describes the orientation from the 0 ...
    frame to the
%                   inertial frame
%   q: a 3x1 vector of generalized coordinates
%   params: a struct of parameters
% Output:
%   C_IC: rotation matrix describing the camera frame orientation
%         with respect to the inertial frame when the base ...
    orientation is
%         not fixed.

  q_I0 = [qW; qX; qY; qZ];

  % implement your solution here ...
  % extract the vector part of the quaternion
  quat_n = q_I0(2:4);
  skew_matrix = [0, -quat_n(3), quat_n(2);
                 quat_n(3), 0, -quat_n(1);
                 -quat_n(2), quat_n(1), 0];
  % convert the quaternion to a rotation matrix
  C_I0 = eye(3,3) + 2*qW*skew_matrix + 2*skew_matrix*skew_matrix;

  % get the matrix C_0C
  T_0C = T_0C_solution(q,params);
  C_0C = T_0C(1:3, 1:3);

```

```
  C_IC = C_I0 * C_0C;

end
```