Exercise 1c: Inverse Kinematics of the ABB IRB 120

Prof. Marco Hutter*
Teaching Assistants: Dario Bellicoso, Jan Carius†

October 9, 2018

Abstract

The aim of this exercise is to calculate the inverse kinematics of an ABB robot arm. To do this, you will have to implement a pseudo-inversion scheme for generic matrices. You will also implement a simple motion controller based on the kinematics of the system. A separate MATLAB script will be provided for the 3D visualization of the robot arm.

Figure 1: The ABB IRW 120 robot arm.

*original contributors include Michael Blösch, Dario Bellicoso, and Samuel Bachmann
†bellicoso@mavt.ethz.ch, jan.carius@mavt.ethz.ch

1 Introduction

The following exercise is based on an ABB IRB 120 depicted in Fig. 1. It is a 6-link robotic manipulator with a fixed base. During the exercise you will implement several different MATLAB functions, which, you should test carefully since the following tasks are often dependent on them. To help you with this, we have provided the script prototypes at `http://www.rsl.ethz.ch/education-students/lectures/robotdynamics.html` together with a visualizer of the manipulator.

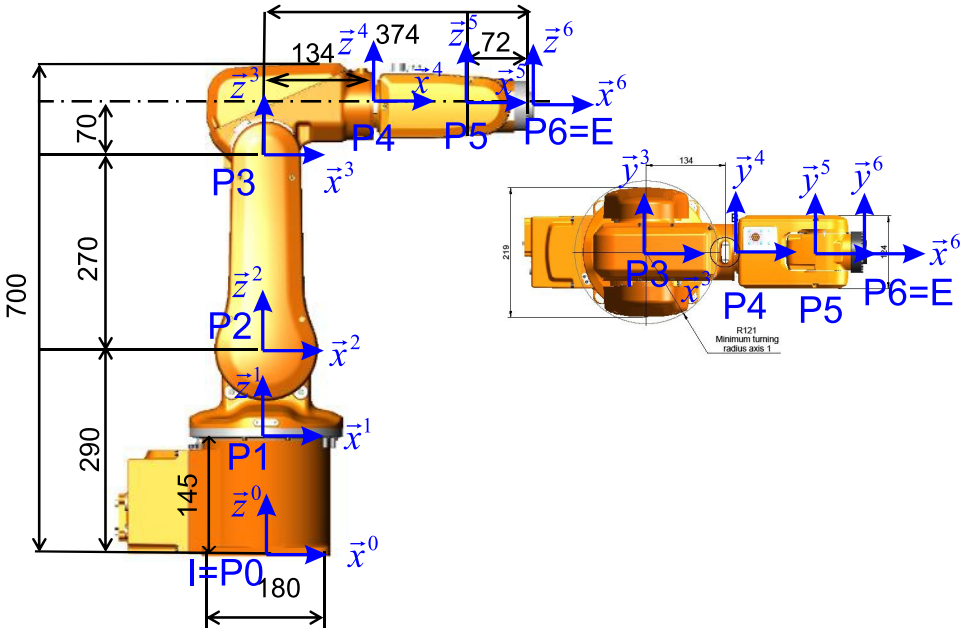

Figure 2: ABB IRB 120 with coordinate systems and joints

Throughout this document, we will employ I for denoting the inertial world coordinate system (which has the same pose as the coordinate system P0 in figure 2) and E for the coordinate system attached to the end-effector (which has the same pose as the coordinate system P6 in Fig. 2).

2 Matrix Pseudo-Inversion

The *Moore-Penrose pseudo-inverse* is a generalization of the *matrix inversion* operation for non-square matrices. Let a non-square matrix A be defined in $\mathbb{R}^{m\times n}$. When $m \geq n$ and $rank(\mathrm{A}) = n$, it is possible to define the so-called *left pseudo-inverse* A_l^+ as

$$\mathrm{A}_l^+ := (\mathrm{A}^T\mathrm{A})^{-1}\mathrm{A}^T, \tag{1}$$

which yields $\mathrm{A}_l^+\mathrm{A} = \mathbb{I}_{n\times n}$. If instead it is $m \leq n$ and $rank(\mathrm{A}) = m$, then it is possible to define the *right pseudo-inverse* A_r^+ as

$$\mathrm{A}_r^+ := \mathrm{A}^T(\mathrm{A}\mathrm{A}^T)^{-1}, \tag{2}$$

which yields $\mathrm{A}\mathrm{A}_r^+ = \mathbb{I}_{m\times m}$. If one wants to handle singularities, then it is possible to define a *damped pseudo-inverse* with damping factor λ as

$$\mathrm{A}_l^+ := (\mathrm{A}^T\mathrm{A} + \lambda^2\mathbf{I}_{n\times n})^{-1}\mathrm{A}^T, \tag{3}$$

and

$$\mathrm{A}_r^+ := \mathrm{A}^T(\mathrm{A}\mathrm{A}^T + \lambda^2 \mathbf{I}_{m\times m})^{-1}. \tag{4}$$

Note that for square and invertible matrices, the pseudo-inverse is equivalent to the usual matrix inverse.

Exercise 2.1

In this first exercise, you are required to provide an implementation of (3) and (4) as a MATLAB function. The function place-holder to be completed is:

Listing 1: `pseudoInverseMat.m`

```
function [ pinvA ] = pseudoInverseMat(A, lambda)
% Input: Any m-by-n matrix.
% Output: An  n-by-m pseudo-inverse of the input according to the ...
    Moore-Penrose formula

% Get the number of rows (m) and columns (n) of A
[m, n] = size(A);

% TODO: complete the computation of the pseudo-inverse.
% Hint: How should we account for both left and right ...
    pseudo-inverse forms?
pinvA = zeros(n, m);
end
```

3 Iterative Inverse Kinematics

Consider a desired position ${}_{\mathcal{I}}\mathbf{r}_{IE}^* = \begin{bmatrix}0.5649 & 0 & 0.5509\end{bmatrix}^T$ and orientation $\mathbf{C}_{IE}^* = \mathbf{I}_{3\times 3}$ which shall be jointly called pose $\boldsymbol{\chi}_e^*$. We wish to find the joint space configuration $\mathbf{q}$ which corresponds to the desired pose. This exercise focuses on the implementation of an iterative inverse kinematics algorithm, which can be summarized as follows:

1. $\mathbf{q} \leftarrow \mathbf{q}^0$ ▷ start configuration
2. while $\|\boldsymbol{\chi}_e^* \boxminus \boldsymbol{\chi}_e(\mathbf{q})\| > tol$ ▷ while the solution is not reached
3. $\mathbf{J}_{e0} \leftarrow \mathbf{J}_{e0}(\mathbf{q})$ ▷ evaluate Jacobian for current $\mathbf{q}$
4. $\mathbf{J}_{e0}^+ \leftarrow (\mathbf{J}_{e0})^+$ ▷ update the pseudoinverse
5. $\Delta\boldsymbol{\chi}_e \leftarrow \boldsymbol{\chi}_e^* \boxminus \boldsymbol{\chi}_e(\mathbf{q})$ ▷ find the end-effector configuration error vector
6. $\mathbf{q} \leftarrow \mathbf{q} + \alpha \mathbf{J}_{e0}^+ \Delta\boldsymbol{\chi}_e$ ▷ update the generalized coordinates (step size α)

Note that we are using the geometric Jacobian $\mathbf{J}_{e0}$, which was derived in the last exercise. The boxminus ($\boxminus$) operator is a generalized difference operator that allows "substraction" of poses. The orientation difference is thereby defined as the rotational vector extracted from the relative rotation between the desired orientation $\mathbf{C}_{IE}^*$ and the one based on the solution of the current iteration $\mathbf{C}_{IE}(\mathbf{q})$, i.e.,

$$\Delta\boldsymbol{\varphi} = {}_I\boldsymbol{\varphi}_{EE*} = rotMatToPhi(\mathbf{C}_{IE}^* \mathbf{C}_{IE}^T(\mathbf{q}))\,. \tag{5}$$

Exercise 3.1

Your task is to implement the iterative inverse kinematics algorithm by completing the following two Matlab functions. Use `rotMatToRotVec` as a helper function to calculate the pose error.

Note: Your implementation should be robust against the case for which the rotation is identity, i.e. the rotation angle is zero.

Note2: You can test the `inverseKinematics` function by calling it with arguments of your choice.

Listing 2: `rotMatToRotVec.m`

```
function [ phi ] = rotMatToRotVec(C)
% Input: a rotation matrix C
% Output: the rotational vector which describes the rotation C

% Compute the rotional vector
phi = zeros(3,1);
end
```

Listing 3: `inverseKinematics.m`

```
function [ q ] = inverseKinematics(I_r_IE_des, C_IE_des, q_0, tol)
% Input: desired end-effector position, desired end-effector ...
    orientation (rotation matrix),
%        initial guess for joint angles, threshold for the ...
    stopping-criterion
% Output: joint angles which match desired end-effector position ...
    and orientation

% 0. Setup
it = 0;
max_it = 100;       % Set the maximum number of iterations.
lambda = 0.001;     % Damping factor
alpha = 0.5;        % Update rate

close all;
loadviz;

% 1. start configuration
q = q_0;

% 2. Iterate until terminating condition.
while (it==0 || (norm(dxe)>tol && it < max_it))
    % 3. evaluate Jacobian for current q
    I_J = ;

    % 4. Update the psuedo inverse
    I_J_pinv = ;

    % 5. Find the end-effector configuration error vector
    % position error
    dr = ;
    % rotation error
    dph = ;
    % pose error
    dxe = ;

    % 6. Update the generalized coordinates
    q = ;

    % Update robot
    abbRobot.setJointPositions(q);
```

```
39      drawnow;
40      pause(0.1);
41
42      it = it+1;
43  end
44
45  % Get final error (as for 5.)
46  % position error
47  dr = ;
48  % rotation error
49  dph = ;
50
51  fprintf('Inverse kinematics terminated after %d iterations.\n', it);
52  fprintf('Position error: %e.\n', norm(dr));
53  fprintf('Attitude error: %e.\n', norm(dph));
54  end
```