Prof. Dr. Maren Bennewitz
M.Sc. Stefan Oßwald
M.Sc. AbdElMoniem Bayoumi

Bonn, 18 May 2017

Humanoid Robots
Exercise Sheet 3 - Forward Kinematics

Exercise 5 (10 points)

The task of computing the end-effector pose given the current joint encoder readings is called *forward kinematics*. In this exercise, we will compute the forward kinematics for the left arm of a Nao robot.

For the kinematic computations, we will use the Denavit-Hartenberg (DH) representation[1]. Each joint is represented by a coordinate system (X, Y, Z) and four parameters (d, a, θ, α). The Z axis is always the rotation axis of the joint. For an explanation of the other axes and parameters, see the following video by Ethan Tira-Thompson: `https://www.youtube.com/watch?v=rA9tm0gTln8`.

The layout of Nao's joints are given on slide 5 of the slides on whole-body self-calibration. The DH parameters of the left arm's kinematic chain are as follows:

joint name	abbrev.	d [meters]	a [meters]	θ [radians]	α [radians]
LShoulderPitch	LSP	0	0	0	$\frac{\pi}{2}$
LShoulderRoll	LSR	0	0	$\frac{\pi}{2}$	$\frac{\pi}{2}$
LElbowYaw	LEY	0.0900	0	0	$-\frac{\pi}{2}$
LElbowRoll	LER	0	0	0	$\frac{\pi}{2}$
LWristYaw	LWY	0.0506	0	0	$\frac{\pi}{2}$

The goal of this exercise is to compute the transformation $\mathcal{F}_E^B(\hat{\mathbf{q}})$ from the left hand frame E to the robot's base frame B given the encoder readings $\hat{\mathbf{q}}$.

This transformation can be factorized to

$$\mathcal{F}_E^B(\hat{\mathbf{q}}) = \mathbf{T}^B_{shoulder} \cdot \mathbf{A}^{shoulder}_{LSP} \cdot \mathbf{A}^{LSP}_{LSR} \cdot \mathbf{A}^{LSR}_{LEY} \cdot \mathbf{A}^{LEY}_{LER} \cdot \mathbf{A}^{LER}_{LWY} \cdot \mathbf{T}^{LWY}_{E}. \tag{1}$$

The point where the arm is attached to the torso is given by the homogeneous transformation

[1] Note that there are multiple conventions for aligning the coordinate systems and for numbering the indices in the literature. We will used the "modified" version of DH and use only one index per transformation.

matrix

$$\mathbf{T}^{B}_{shoulder} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.098 \\ 0 & -1 & 0 & 0.100 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

and the point where the hand is attached to the other end of the arm is given by

$$\mathbf{T}^{LWY}_{E} = \begin{pmatrix} 0 & 1 & 0 & 0.0159 \\ 1 & 0 & 0 & 0.0580 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3}$$

Each transformation $\mathbf{A}_i^{i-1}(q_i)$ consists of four coordinate transformations:

$$\mathbf{A}_i^{i-1}(q_i) = \mathrm{Rot}_z\left(\theta_i + q_i\right) \cdot \mathrm{Trans}_z\left(d_i\right) \cdot \mathrm{Rot}_x(\alpha_i) \cdot \mathrm{Trans}_x\left(a_i\right) \tag{4}$$

where $\mathrm{Rot}_z, \mathrm{Rot}_x, \mathrm{Trans}_z, \mathrm{Trans}_x$ are the homogeneous matrices for rotation around the z axis, rotation around the x axis, translation along the z axis, and translation along the x axis, respectively. See the slides on homogeneous coordinates for the definitions of the matrices.

a) Implement the homogeneous rotation and translation matrices for the given axes in `rotationX`, `rotationZ`, `translationX`, and `translationZ`.

b) Implement the method `getA` for computing $\mathbf{A}_i^{i-1}(q_i)$ according to Eq. (4).

c) Implement the method `computeHandTransform` for computing the complete transformation $\mathcal{F}_E^B(\hat{\mathbf{q}})$ between the robot's torso and the hand end-effector according to Eq. (1).

If you have Gnuplot installed on your computer, then you can execute the file `scripts/plot.gp` to get an interactive 3D view of the resulting trajectory (Windows: double-click the file in explorer / Linux: change with `cd` to the `scripts` directory and run `./plot.gp`).

On the Wiki, you'll find the same plot and also a video of the robot's motion in a simulation environment.

Exercise 6 (10 points)

This exercise is a pen-and-paper exercise without source code. Hand in your solution by putting a PDF file named `src/06_kdtree/solution.pdf` in your GIT repository.

A *kd-tree* is an efficient data structure for storing 3D points (or even higher-dimensional tuples). In kd-trees, location queries can be executed in time $\mathcal{O}(\log n)$, compared to $\mathcal{O}(n)$ in the case of ordinary lists.

The kd-tree is a special case of binary trees, but for k instead of two dimensions. There are many ways to select the root node and pivot elements. The pivot elements are the nodes of the tree which determine the splitting planes.

In this exercise, we are considering the simple, balanced kd-trees. In this method, we select one of the available dimensions and sort the set of points based on the value of that selected dimension. Then, we select the median point as the pivot element (without replacement from the set of points). Based on that, the set of points is divided into two sets: one set contains the points of the left sub-tree which have smaller values compared to the pivot point (relative to the selected dimension), and the other set contains the remaining points. After that, this step will be repeated recursively for every set of points and we keep cycling through the different available dimensions throughout the tree levels.

Generate a kd-tree for the following set of 3D points:

$$\begin{bmatrix} 5 & 8 & 2 & -8 & -7 & 5 & 7 & 3 & -4 & 9 & -1 & 0 & 0 & 2 & 4 & 9 & 10 \\ 0 & -1 & -10 & 5 & 7 & 7 & 3 & 2 & -1 & 1 & 5 & 7 & 5 & 4 & 7 & -8 & -3 \\ -3 & -4 & 8 & 6 & -6 & 15 & 4 & 2 & 3 & -4 & 7 & 0 & 2 & -1 & 4 & 5 & -7 \end{bmatrix}$$

Each column represents a point in the form of (x, y, z) (e.g., $(x_1, y_1, z_1) = (5, 0, -3)$).

Exercise steps:

a) Generate the kd-tree for the given set of points and draw it.

b) Save your drawn tree to a file named as `src/06_kdtree/solution.pdf` (Note: hand-drawn figures are acceptable as well as computer-generated ones).

c) Git add, commit and push that file to the exercise workspace.

Deadline: 26 May 2017, 11:59 am