Robotics
Exercise 3

Marc Toussaint
Lecturer: Duy Nguyen-Tuong
TAs: Philipp Kratzer, Janik Hager, Yoojin Oh
Machine Learning & Robotics lab, U Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany

November 6, 2018

1 Motion profiles (2 points)

Construct a motion profile that accelerates constantly in the first quarter of the trajectory, then moves with constant velocity, then decelerates constantly in the last quarter. Write down the equation $\text{MP}(s) : [0,1] \mapsto [0,1]$.

2 Verify some things from the lecture (6 points)

a) On slide 02:30 we derived the basic inverse kinematics law. Verify that (assuming linearity of ϕ, i.e., $J\delta q = \delta y$) for $C \to \infty$ the desired task is fulfilled exactly, i.e., $\phi(q^*) = y^*$. By writing $C \to \infty$ we mean that C is a matrix of the form $C = \varrho C_0$, $\varrho \in \mathbb{R}$, and we take the limit $\varrho \to \infty$.
b) On slide 02:36 there is a term $(\mathbf{I} - J^\# J)$ called nullspace projection. Verify that for $\varrho \to \infty$ (and $C = \varrho \mathbf{I}$) and any choice of $\delta a \in \mathbb{R}^n$

$$\delta q = (\mathbf{I} - J^\# J)\delta a \ \Rightarrow\ \delta y = 0$$

(assuming linearity of ϕ, i.e., $J\delta q = \delta y$). Note: this means that any choice of δa, the motion $(\mathbf{I} - J^\# J)\delta a$ will not change the "endeffector position" y. (2 P)
c) On slide 02:48 it says that

$$\begin{aligned}&\operatorname*{argmin}_q \|q - q_0\|^2_W + \|\Phi(q)\|^2 \\ &\quad\approx q_0 - (J^\top J + W)^{-1} J^\top\, \Phi(q_0) = q_0 - J^\# \Phi(q_0)\end{aligned}$$

where the approximation $\approx$ means that we use the local linearization $\Phi(q) = \Phi(q_0) + J(q - q_0)$. Verify this. (2 P)

3 IK in the simulator (6 points)

Installation instructions:

1. On github `https://github.com/MarcToussaint/robotics-course` you can find the course repository and an instruction on how to install it.
2. To make sure you have an updated version of the repository, run '*git pull*' and '*git submodule update*'
3. For python you can run: *'jupyter-notebook py/01-kinematics/01-kinematics.ipynb'*
4. For C++ run: *'cd cpp/01-kinematics'*, *'make'*, *'./x.exe -mode 2'*

The goal of this task is to reach the coordinates $y^* = (-0.2, -0.4, 1.1)$ with the right hand of the robot. Assume $W = \mathbf{I}$ and $\sigma = .01$.

a) The provided code already generates a motion using inverse kinematics $\delta q = J^\sharp\ \delta y$ with $J^\sharp = (J^\top J + \sigma^2 W)^{-1} J^\top$. Record the task error, that is, the deviation of hand position from y^* after each step. You can plot the error using *'plt.plot(err)'* and *'plt.show()'* in python or *'gnuplot(err)'* in C++ (*err* is the array of errors). Why is it initially large? (1 P)

b) Try to do 100 smaller steps $\delta q = \alpha J^\sharp\ \delta y$ with $\alpha = .1$ (each step starting with the outcome of the previous step). How does the task error evolve over time? (1 P)

c) Generate a nice trajectory composed of $T = 100$ time steps. Interpolate the target linearly $\widehat{y} \leftarrow y_0 + (t/T)(y^* - y_0)$ in each time step. How does the task error evolve over time? (2 P)

d) Generate a trajectory that moves the right hand in a circle centered at $(-0.2, -0.4, 1.1)$, aligned with the xz-plane, with radius 0.2. (2 P)