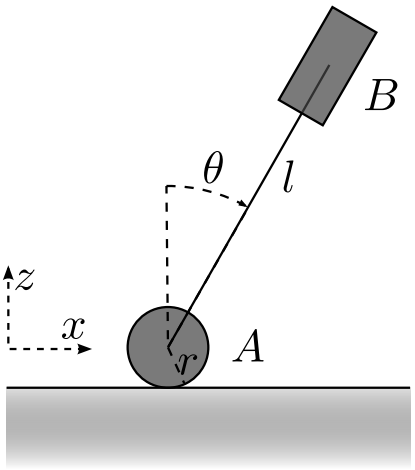
Robotics
Exercise 4

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 12, 2019

1 Fun with Euler-Lagrange (6 points)



Consider an inverted pendulum mounted on a wheel in the 2D x-z-plane; similar to a Segway. The exercise is to derive the Euler-Lagrange equation for this system. Strictly follow the scheme we discussed on slide 03:23.

a) Describe the **pose** p_i of every body (depending on q) in (x, z, ϕ) coordinates: its position in the x-z-plane, and its rotation ϕ relative to the world-vertical. (2P)

b) Describe the (linear and angular) velocity v_i of every body. (1P)

c) Formulate the kinetic energy T. (1P)

d) Formulate the potential energy U. (1P)

e) Compute the Euler-Lagrange Equation: (1P)

$$\tau = \frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q}$$

Tips:

- Use the generalized coordinates

$$q = (x, \theta) \tag{1}$$

  where x is the position of the wheel and θ the angle of the pendulum relative to the world-vertical.

- The system can be parameterized by
  - m_A, I_A, m_B, I_B: masses and inertias of bodies A (=wheel) and B (=pendulum)
  - r: radius of the wheel
  - l: length of the pendulum (height of its COM)

- In this 3-dim space, the mass matrix of every body is

$$M_i = \begin{pmatrix} m_i & 0 & 0 \\ 0 & m_i & 0 \\ 0 & 0 & I_i \end{pmatrix} \tag{2}$$

(this allows to compute the kinetic energy of a body i with $\frac{1}{2} v_i^\top M_i v_i$)

2 Multiple task variables (6 points)

1. To make sure you have an updated version of the repository, run '*git pull*' and '*git submodule update*'
2. For python you can run: *'jupyter-notebook py/02-kinematics/02-kinematics.ipynb'*
3. For C++ run: *'cd cpp/02-kinematics'*, *'make'*, *'./x.exe -mode 4'*
4. To understand how to access and use the code, refer to https://github.com/MarcToussaint/robotics-course/blob/master/docs/2-features.ipynb

The code contains the solution to last week's exercise where the robot moved his hand in a circle.

With multiple tasks, you can stack each objective and its Jacobian to Phi and PhiJ using *Phi.append(objective)* (C++) or *np.vstack((Phi, objective))* (Python)

a) We've seen that the solution does track the circle nicely, but the trajectory "gets lost" in joint space, leading to very strange postures. We can fix this by adding more tasks, esp. a task that permanently tries to (moderatly) minimize the distance of the configuration q to a natural posture q_{home}. Realize this by adding a respective task. (2P)

b) Make the robot simultaneously point upward with the left hand. Use the following commands to compute the orientation of a robot shape for this. (2P)

For C++:

```
arr y, J;
K.evalFeature(y, J, FS_vectorZ, {"handL"});
```

For Python:

```
F = K.feature(lry.FS.vectorZ, ["handL"])
y, J = F.eval(K)
```

c) Use the task spaces from Exercise 02 question 2.c) to add a task such that the robot looks with his right eye towards the right hand. You can use the following task spaces and the derivative to calculate the Jacobian. (2P)

$$\phi_1(q) = \phi^{\text{vec}}_{h,g} - \frac{x^W - \phi^{\text{pos}}_{h,e}}{|x^W - \phi^{\text{pos}}_{h,e}|} \quad \in \mathbb{R}^3$$

$$\phi_2(q) = \phi^{\text{vec}\top}_{h,g} \Big[\frac{x^W - \phi^{\text{pos}}_{h,e}}{|x^W - \phi^{\text{pos}}_{h,e}|} \Big] - 1 \quad \in \mathbb{R}^1$$

$$\begin{aligned} \partial_x \frac{f(x)}{|f(x)|} &= \partial_x \frac{f(x)}{[f(x)^\top f(x)]^{\frac{1}{2}}} \\ &= \frac{1}{[f(x)^\top f(x)]^{\frac{1}{2}}} \partial_x f(x) + f(x) \partial_x \frac{1}{[f(x)^\top f(x)]^{\frac{1}{2}}} \\ &= \frac{1}{[f(x)^\top f(x)]^{\frac{1}{2}}} \partial_x f(x) + f(x) \frac{-1/2}{[f(x)^\top f(x)]^{3/2}} [2 f(x)^\top \partial_x f(x)] \\ &= \frac{1}{|f(x)|} \Big[\mathbf{I} - \frac{f(x) f(x)^\top}{f(x)^\top f(x)} \Big] \partial_x f(x) \end{aligned}$$

You can use the following frame to access the right eye.

For C++:

```
K.evalFeature(y, J, FS_vectorZ, {"eyeR"});
```

For Python:

```
F = K.feature(lry.FS.vectorZ, ["eyeR"])
y, J = F.eval(K)
```