Optimization, Robotics & Biomechanics (ORB)
Institute of Computer Engineering (ziti)
Heidelberg University

Prof. Dr. Katja Mombaur,
Benjamin Reh,
Silvan Lindner

Robotics 1 (WS 2018/2019)

Exercise Sheet 3

Presentation during exercises in calendar week 47

Exercise 3.1 – Degrees of Freedom

While the number of degrees of freedom of kinematic chains or similar can often be calculated straight forward by simply adding the degrees of freedom of all joints, this can quickly become more difficult if we are dealing with kinematic loops or kinematic graphs in general. Fortunately the number of degrees of freedom of a mechanism with links and joints can conveniently be calculated using Grübler's formula:

Grübler's Formula: Consider a mechanism consisting of N links, where ground is also regarded as a link. Let J be the number of degrees of freedom of a rigid body ($m = 3$ for planar mechanism and $m = 6$ for spatial mechanism), f_i be the number of freedoms provided by joint i and c_i be the number of constraints provided by joint i, where $f_i + c_i = m$ for all i. Then Grübler's formula for the number of degrees of freedom of the robot is

$$dof = m(N - 1 - J) + \sum_{i=1}^{J} f_i.$$

Note: This formula holds only if all joint constraints are independent. If they are not independent then the formula provides a lower bound on the number of degrees of freedom.

- Familiarize yourself with Grübler's formula.
- Use the appropriate version of Grübler's formula to determine the number of degrees of freedom for all mechanisms in figure 1. Comment on whether your results agree with your intuition about the possible motions of these mechanisms.
  Hint: Treat each arm in figure 1d as a new joint type (The arms wont add to the total link count anymore!).
- Suppose there are now n instead of 4 arms in the mechanism from figure 1d. How many degrees of freedom does this system have?

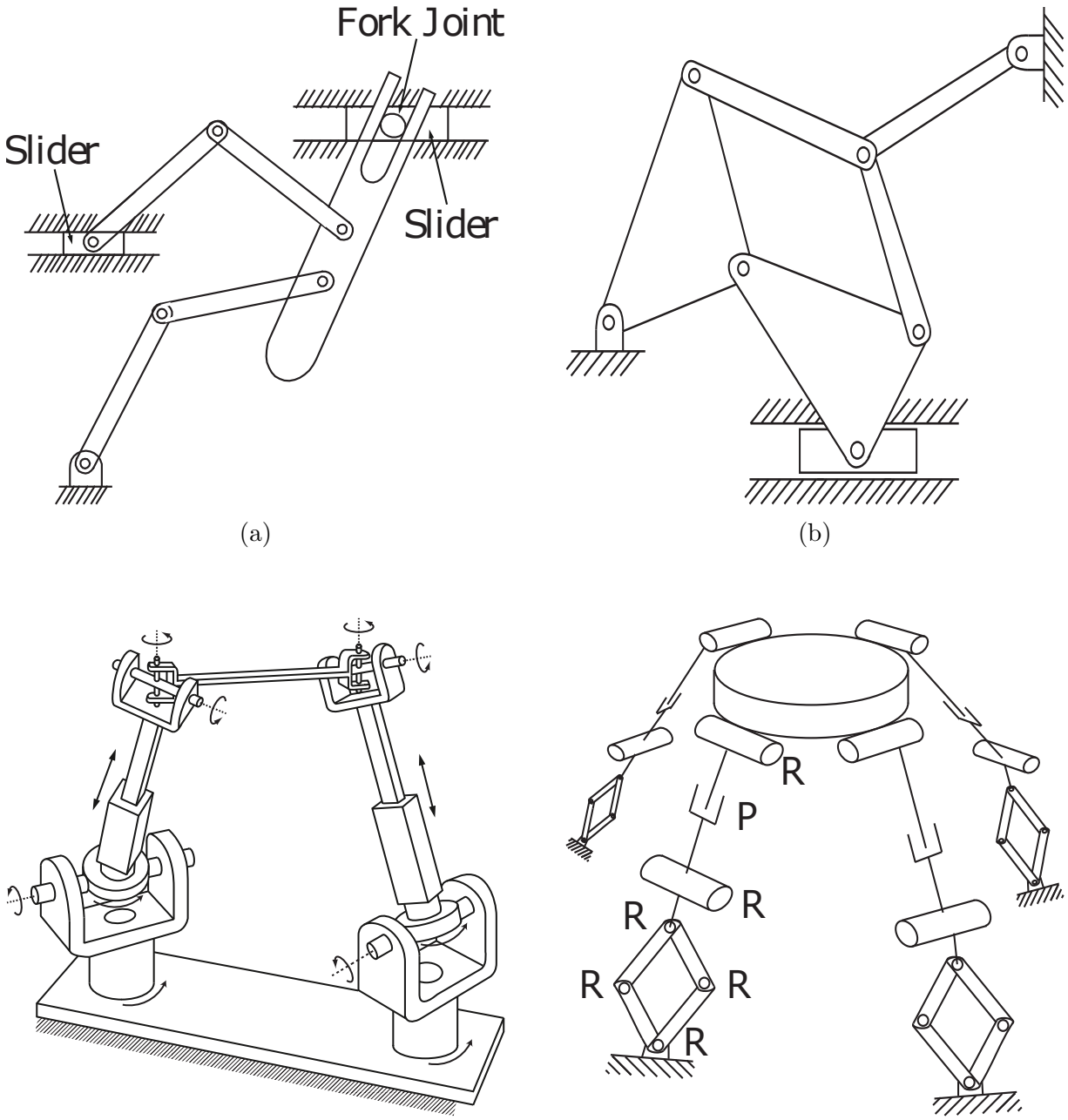


Figure 1: Collection of mechanisms. (R = Revolute joint, P = prismatic joint)

Exercise 3.2 – Angle-Axis Rotations

You have already learned about Euler angles in the lecture, three angles that describe the orientation of a rigid body with respect to a fixed coordinate system. While there are many other ways to represent rotations, they are often converted to either $SO(3)$ (rotation matrices), where

$$SO(3) = \left\{ R \in \mathbb{R}^3 | R^T R = I, \det(R) = 1 \right\},$$

or quaternions, which are computational faster and numerical more stable, in order to concatenate multiple transformations.

A big problem of Euler angles is, that they have a singularity for certain angles:

- Calculate the rotation matrix $R = R_x(\alpha)R_y(\beta)R_z(\gamma)$ for the Euler angles (α, β, γ) at $\beta = \frac{\pi}{2}$. What do you observe? Why is this a problem?

Additionally rotations described by Euler angles or rotation matrices are non intuitive to construct or interpret by hand. A more intuitive way of defining rotations is given by an angle-axis representations. All rotations can in fact be uniquely defined by a vector $v \in \mathbb{R}^3$ that describes a rotation axis, while $\|v\|_2$ describes the rotation angle. The conversion from angle-axis form to rotation matrix is achieved through Rodrigues' rotation formula:

Rodrigues' Rotation Formula: The rotation matrix defined by angle-axis vector $v \in \mathbb{R}^3$ is given by

$$\exp([v]_\times) = I + \frac{\sin(\|v\|_2)}{\|v\|_2}[v]_\times + \left(\frac{1 - \cos(\|v\|_2)}{\|v\|_2^2}\right)[v]_\times^2, \qquad \text{with } [v]_\times = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix},$$

Note: The resulting rotation matrices are well defined for all v, as the limit $\lim_{\|v\| \to 0} \exp([v]_\times)$ is finite.

- Proof that the matrices $\exp([v]_\times)$ indeed define rotation matrices (e.g. $R^T \cdot R = I$)

  For this exercise you do not have to show, that $\det(\exp([v]_\times)) = 1$.

- The two vectors $v_1, v_2 \in \mathbb{R}^3$ are related by

  $$v_2 = \exp([w]_\times) \cdot v_1$$

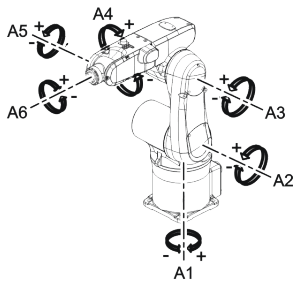
  where $w = \theta \cdot \hat{w} \in \mathbb{R}^3$ is split into axis and angle component ($\|\hat{w}\|_2 = 1$). Given $\hat{w} = \begin{pmatrix} \frac{2}{3} & \frac{2}{3} & \frac{1}{3} \end{pmatrix}^T$, $v_1 = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}^T$, $v_2 = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}^T$, find all the angles $\theta \in [-\pi, \pi]$ that satisfy the above equation.

Additional Information: The group $\mathfrak{so}(3) = \{[e_1]_\times, [e_2]_\times, [e_3]_\times\}$ defines a lie algebra. Since there exists a canonical isomorphism $\mathbb{R}^3 \to \mathfrak{so}(3); v \mapsto [v]_\times$, this lie algebra is especially attractive when optimizing over orientations, since no additional constraints are needed and the number of degrees of freedom matches the number of variables. Even for quaternions q this is not the case, as they requires the additional constraints $\|q\| \overset{!}{=} 1$.

Exercise 3.3 – Forward kinematic

Figure 2a shows the small industrial robot `KR 5 sixx R850`. Figure 3 shows its kinematic structure in the initial position, e.g. all joint angles are set to zero ($q_i = 0$).

(a) Kuka robot `KR 5 sixx R850`



(b) Joints and direction of rotation for `KR 5 sixx R650`

Figure 2: Kuka robots

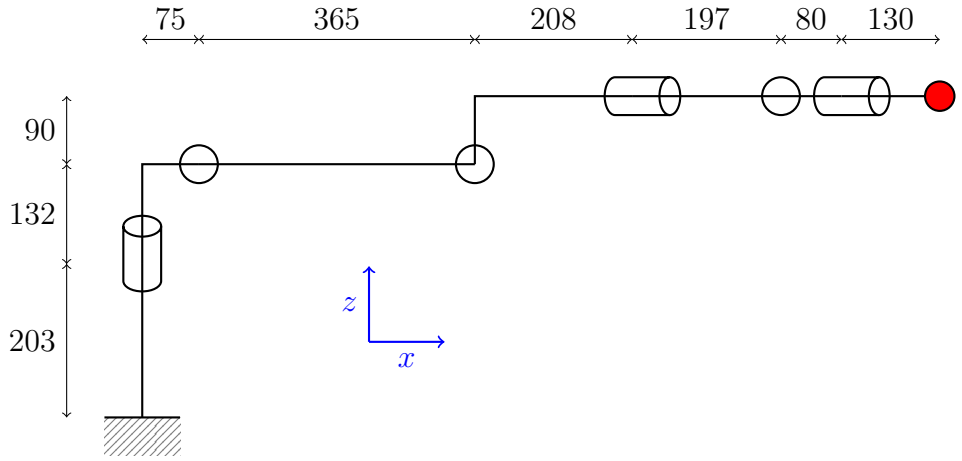


Figure 3: Kinematic chain, lengths are in mm

1. The TCP[1] position can be formulated as a series of translations by vector t_i and rotations defined by Matrix R_i around the joint axes

$$\text{TCP} = t_0 + R_0(t_1 + R_1(t_2 + R_2(\dots))) \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{1}$$

   Express the position of the second joint depending on the angle of rotation of the first joint q_0. Read figure 3 to obtain the segment lengths. Figure 2b shows a similar robot. Use it as a reference for the direction of rotation.

[1] Tool Center Point

What is the position of the second joint for $q_0 = \frac{\pi}{2}$?

What does the vector with zeroes in equation (1) stand for? What is the meaning of the computation if this vector is different?

2. Download, unpack and build the `kuka_kinematics` code snippet. Unpack the `kuka_model.zip` in the same build folder so you are able to start both the compiled program and meshup from the same directory. Start meshup and load the Kuka model and the given `animation.txt` file.
Hint: You can use the meshup command line parameter to load both with just one command:

```
meshup [modelfile [animationfile]]
```

You should see the robot performing a motion while the red ball stays in one place. Take a look at the `animation.txt` file. As you can see there is no MeshUp header as in the excercise on the previous sheet. There are only cloumns of numbers separated by a comma and a blank space. If no header is given, MeshUp assumes the first column to be the time coloumn and the following the joint values in the order given in the lua model file.
In this example each colum means $[t, q_0, q_1 \ldots, q_5, b_x, b_y, b_z]$, where b is the position of the red ball. Run the given code. It reads the `animation.txt` file and writes a file called `animation_with_tcp.txt`.

Implement the forward kinematics for the TCP of the robot inside the following function:

```
Vector3d forwardKinematic(Vector6d q) { }
```

The TCP is visualized as red ball. It is good practise to consecutively build up the chain: start by putting the red ball in joint 0 and then joint 1 etc. up to the TCP. In the end the red ball should follow the TCP position throughout the entire motion.

Hint: The Eigen3 library offers the method `AngleAxisd(angle_in_rad, Vector3d(ax,ay,az) )` that allows to define rotations in angle-axis-representation. The axis vector must be normalized