Kinematik

Gelenkkinematik

- Ziel: Beschreibung der unterschiedlichen Gelenktypen und deren Einfluss auf die Bewegung der Körper

- Definition: Ein (kinematisches) Gelenk ist eine Verbindung zwischen *zwei Segmenten*, die die *relative Bewegung* der beiden Segmente *einschränkt*

- Idealisierungen:
  1. Idealisierte Körper
     - Perfekt starre Körper (keine Deformierung)
     - Ideale Kontaktflächen
     - Ideale Form
  2. Ideale Gelenke
     - Kein Spiel in den Gelenken
     - Keine Flexibilität

- Man spricht von *lower pair joints* (Kontaktflächen) und *higher pair joints* (Kontaktpunkte oder Kontaktlinien)

Lower pair joints

- Man kann beweisen, dass es nur sechs verschieden Arten von *lower pair joints* gibt. Diese sind:

    1. Rotatorisches Gelenk

    2. Prismatisches Gelenk

    3. Spiralgelenk

    4. Zylindrisches Gelenk

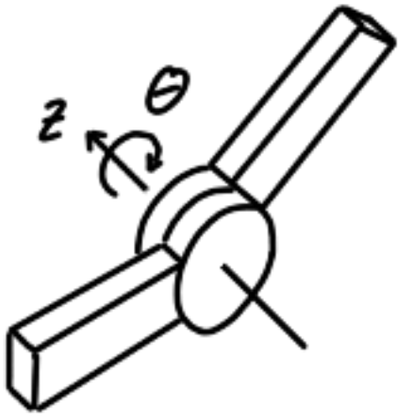
    5. Kugelförmiges Gelenk

    6. Planares Gelenk

Rotationsgelenk (revolute joint)

- Wird mit “R” abgekürzt und oft auch als *hinge joint* (Scharniergelenk) bezeichnet (1 Degree of Freedom)

- Erlaubt die Rotation eines Segments um *eine* Achse des anderen Segments

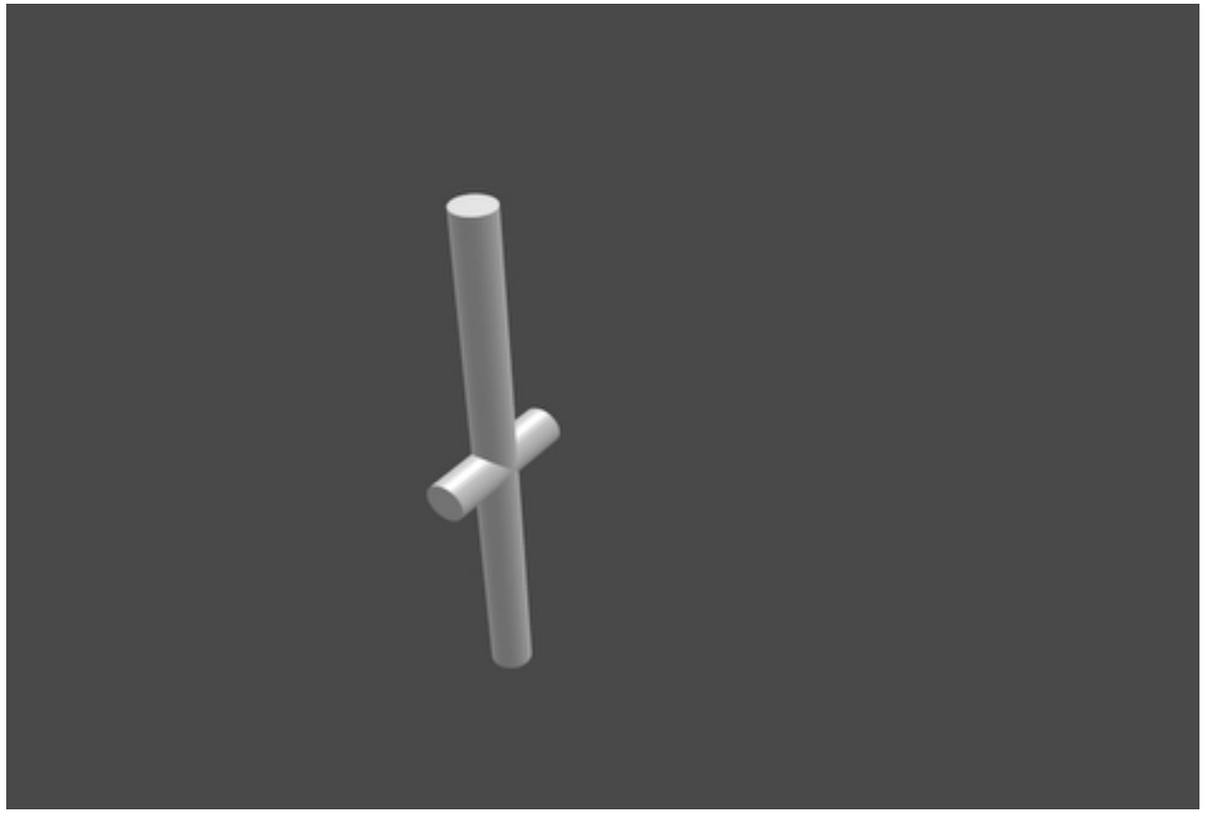


Rotationsmatrix

$$\begin{pmatrix} c_\theta & -s_\theta & 0 \\ s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Positionsvektor

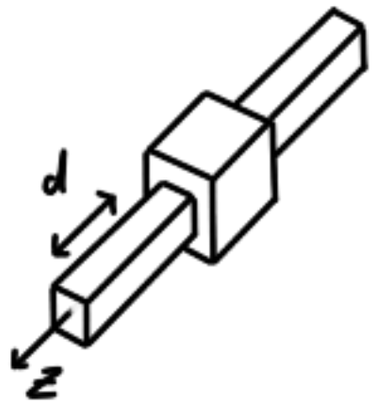
$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$



Prismatisches Gelenk (prismatic joint)

- Wird mit “P” abgekürzt und auch *sliding joint* genannt (Lineares Gelenk)

- Erlaubt die Translation eines Segments relativ zu einem andere Segment entlang nur einer Achse (1 DOF)

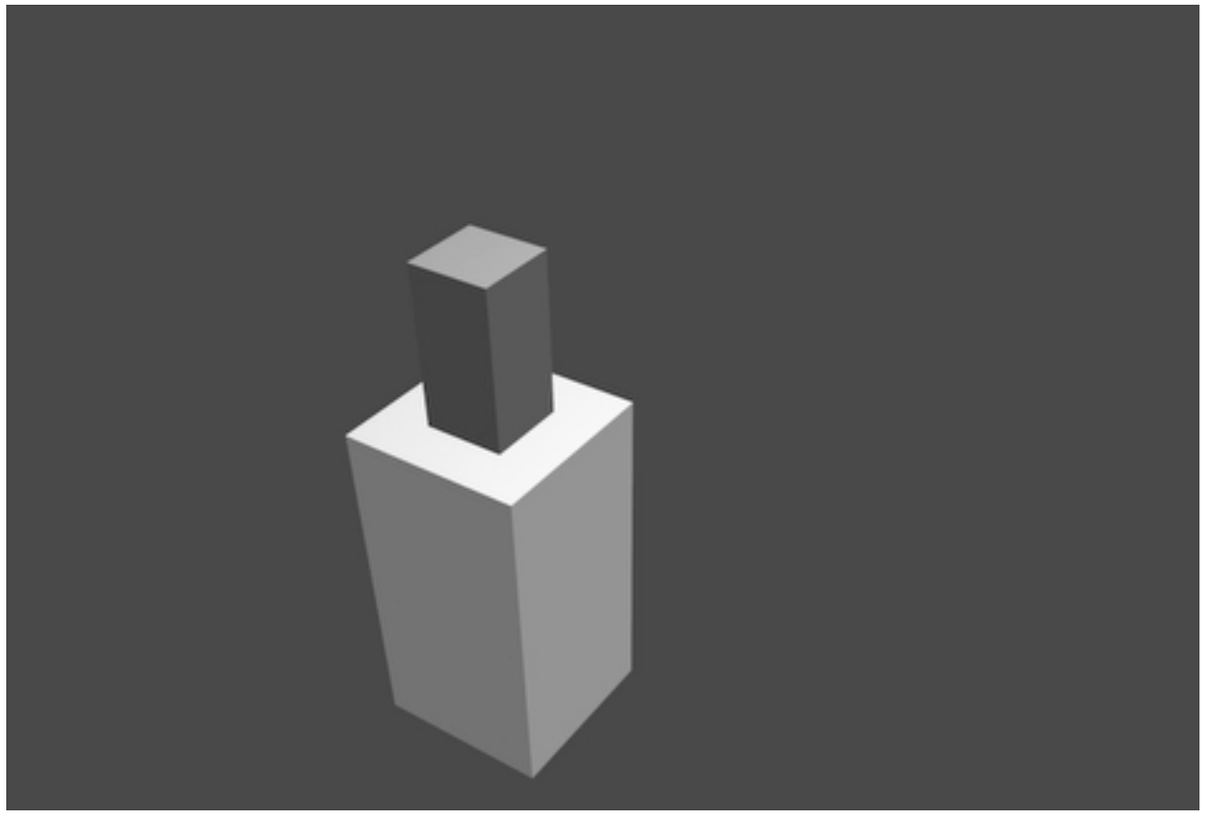


Rotationsmatrix

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

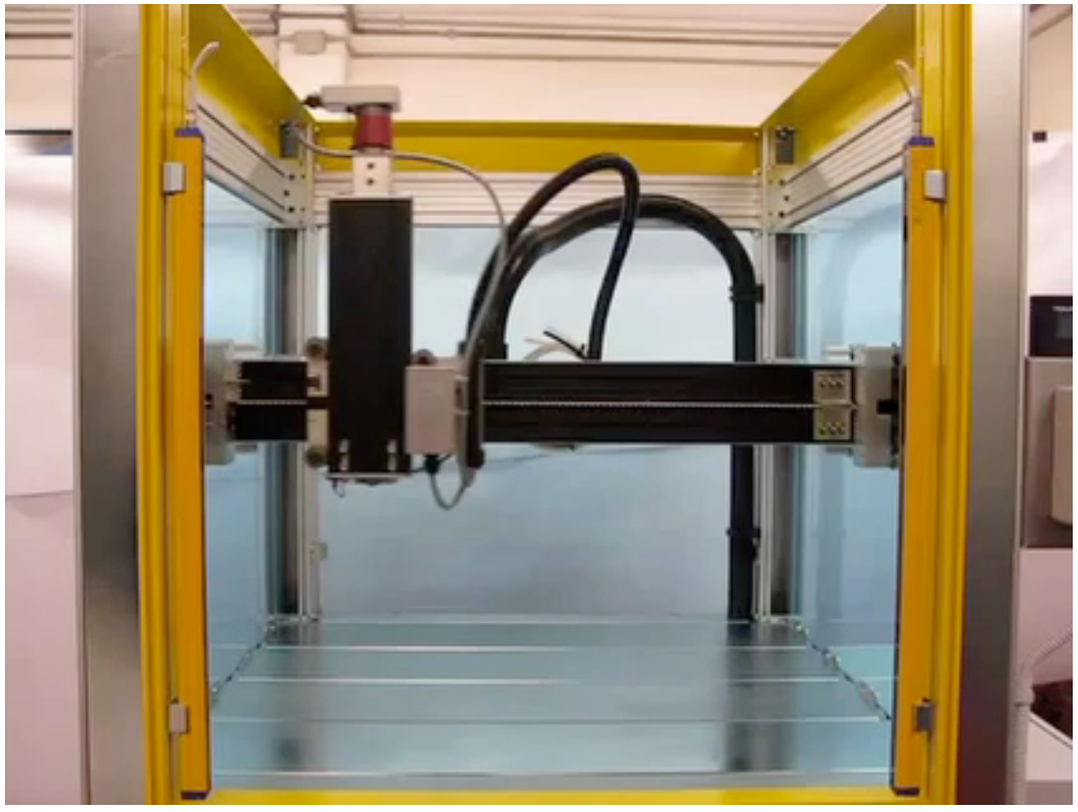
Positionsvektor

$$\begin{pmatrix} 0 \\ 0 \\ d \end{pmatrix}$$



Revolute & prismatic joint — Beispiele

https://www.youtube.com/watch?v=SOESSCXGhFo

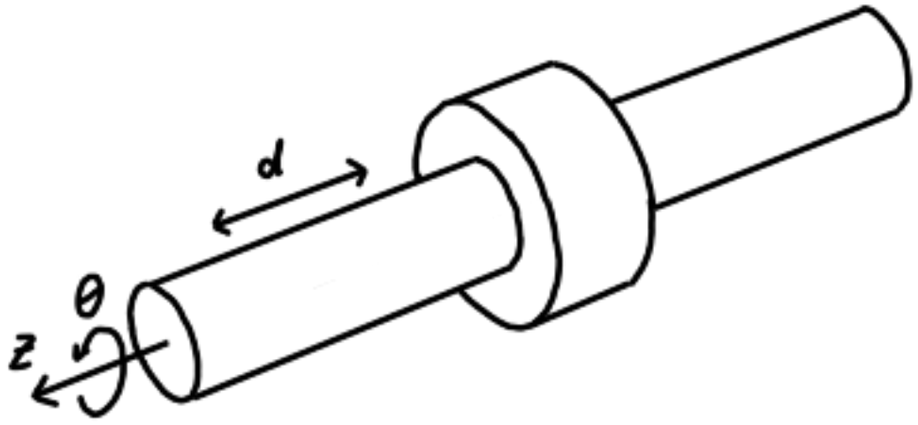


https://www.youtube.com/watch?v=JrP3kl7hV1w

Zylindrische Gelenk (cylindrical joint)

- Wird mit “C” abgekürzt und erlaubt die Translation unabhängig von der Rotation um eine Achse (2 DOF)

- Alle Gelenke mit mehr als 1 DOF kann man aus 1 DOF Gelenken zusammensetzen

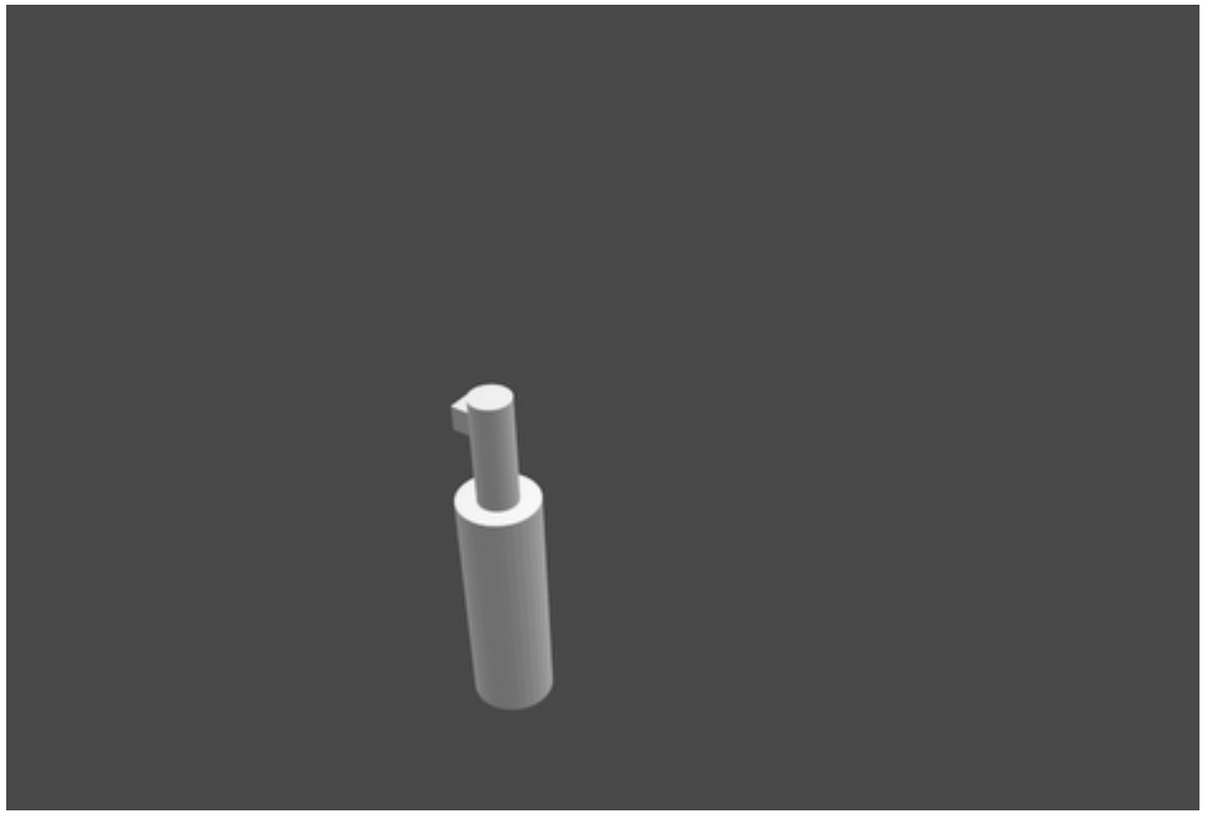


Rotationsmatrix

$$\begin{pmatrix} c_\theta & -s_\theta & 0 \\ s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Positionsvektor

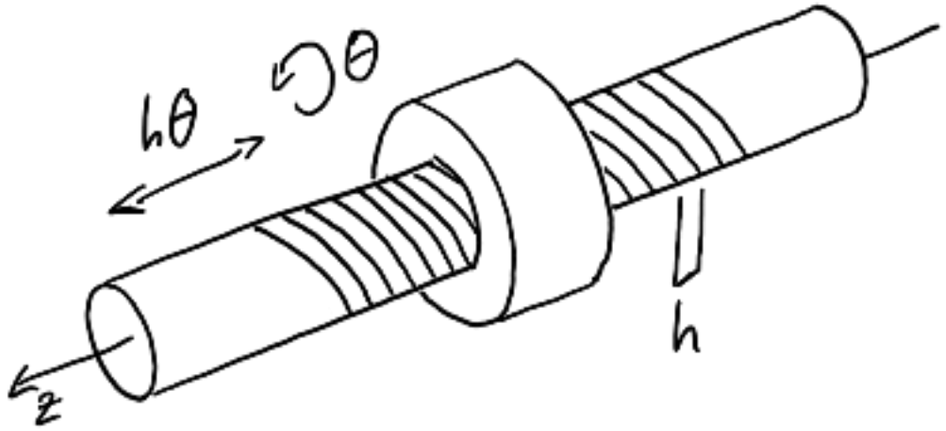
$$\begin{pmatrix} 0 \\ 0 \\ d \end{pmatrix}$$



Spiralgelenk (helical joint)

- Wird mit “H” abgekürzt und auch als *screw joint* (Schraubengelenk) bezeichnet

- Die Rotation um θ ist direkt verbunden mit der Translation d (1 DOF).
  Es gilt $d=h\theta$, h — Gewindesteigung (pitch)

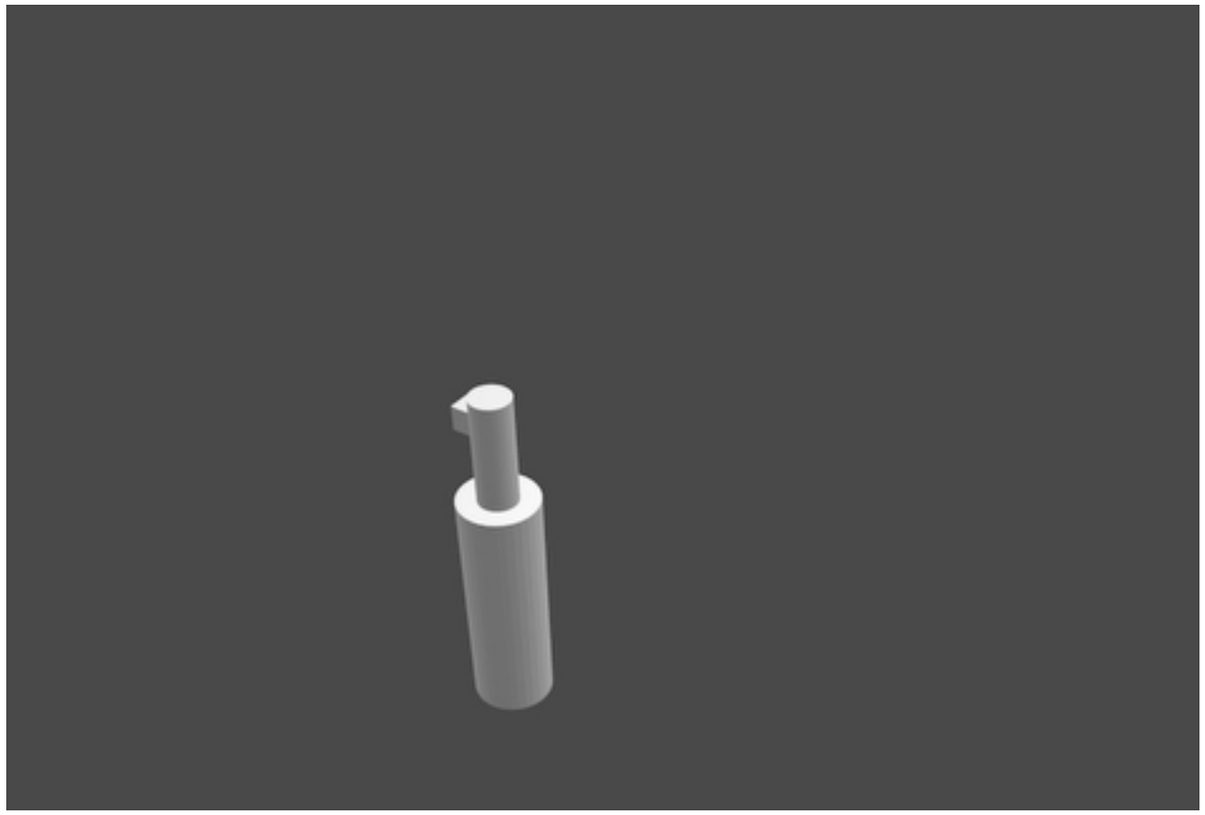


Rotationsmatrix

$$\begin{pmatrix} c_\theta & -s_\theta & 0 \\ s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Positionsvektor

$$\begin{pmatrix} 0 \\ 0 \\ h\theta \end{pmatrix}$$



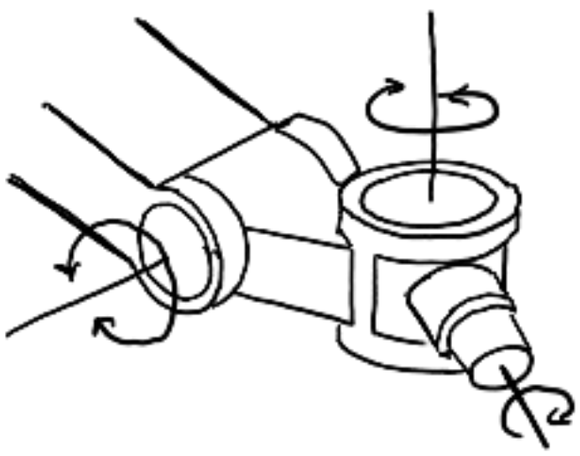
- orthogonale

Kugelgelenk (spherical joint)

- orthogonale

- Wird mit “S” abgekürzt und erlaubt die Rotation um jede beliebige Achse (3 DOF)

- Kann z.B. durch 3 orthogonale, rotatorische Gelenke realisiert werden



Rotationsmatrix

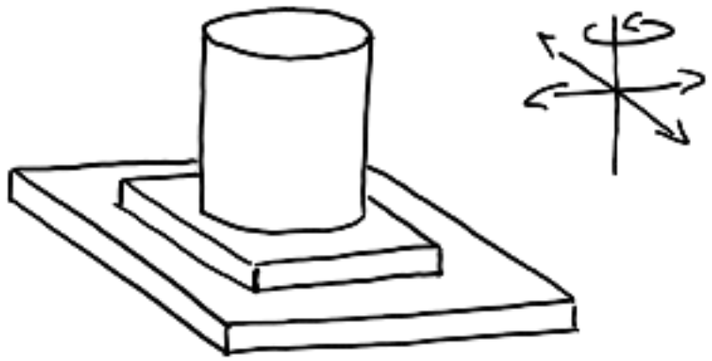
$$\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

Positionsvektor

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Planares Gelenk (planar joint)

- Gelenkt zwischen zwei planar aufeinander liegenden Flächen (3 DOF)
- Kann z.B. durch drei rotatorische Gelenke mit parallelen Achsen erzeugt werden

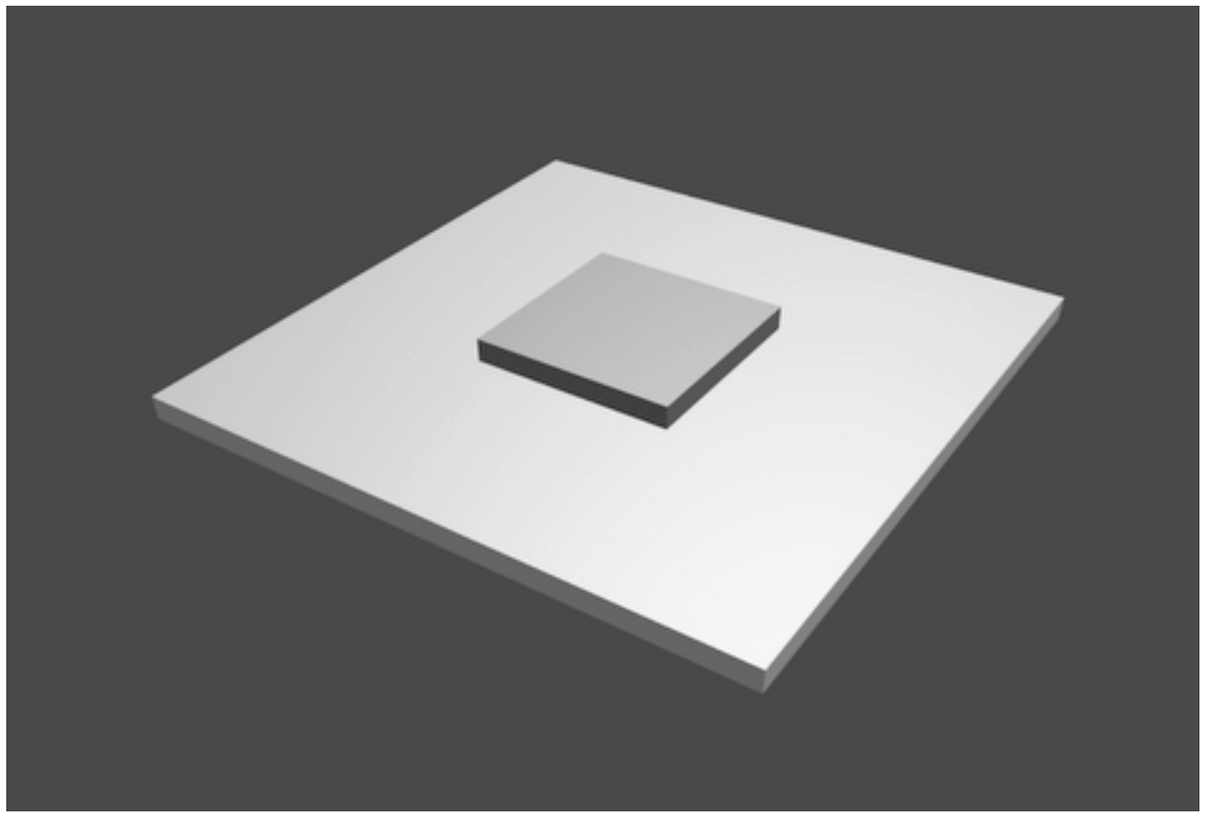


Rotationsmatrix

$$\begin{pmatrix} c_\theta & -s_\theta & 0 \\ s_\theta & c_\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Positionsvektor

$$\begin{pmatrix} c_\theta d_x - s_\theta d_y \\ s_\theta d_x + c_\theta d_y \\ 0 \end{pmatrix}$$



Higher Pair Joint — 6DOF Joint

- Erlaubt die Rotation und Translation unabhängig um jede Achse

- Wird benutzt um ein Gelenk zwischen zwei Segmenten zu modellieren, die nicht mit einander verbunden sind

- Dies wird bei der Modellierung von kinematischen Ketten benötigt

Rotationsmatrix

$$\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

Positionsvektor

$$\begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}$$

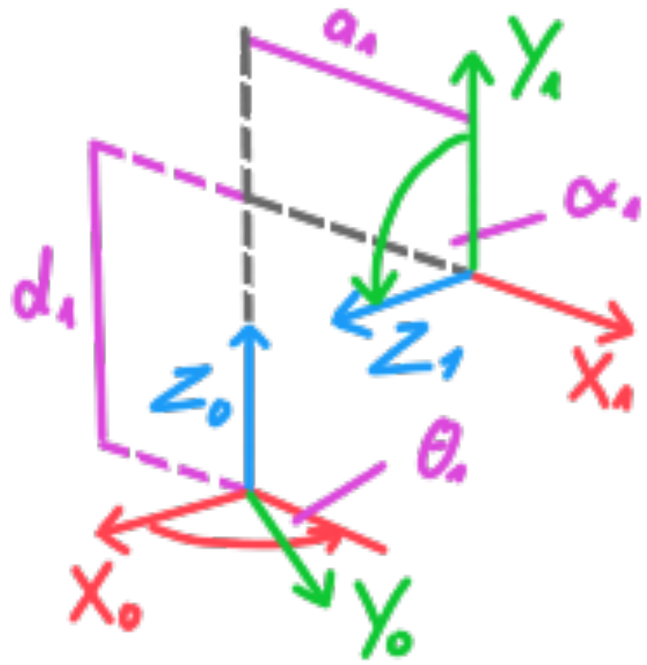
- Jetzt haben wir alles was wir brauchen um die Geometrie eines beliebigen Roboters zu beschreiben

Geometrischen Beschreibung eine Roboters

- Allgemein braucht zur Beschreibung von *N* Körpern 6*N* Variablen
- Durch Konvention und Regeln lassen sich diese auf 4*N* reduzieren
- Die am häufigsten verwendete ist die *Denavit-Hartenberg Konvention*

- Alle Gelenktypen werden durch *revolute* und *prismatic joints* modelliert

- Die vier Parameter sind:
    1. a — Segmentlänge
    2. α — Segmentwinkel
    3. d — Gelenkoffset
    4. θ — Gelenkwinkel



Denavit-Hartenberg Regeln

<u>Schritt 1:</u> Nummerierung

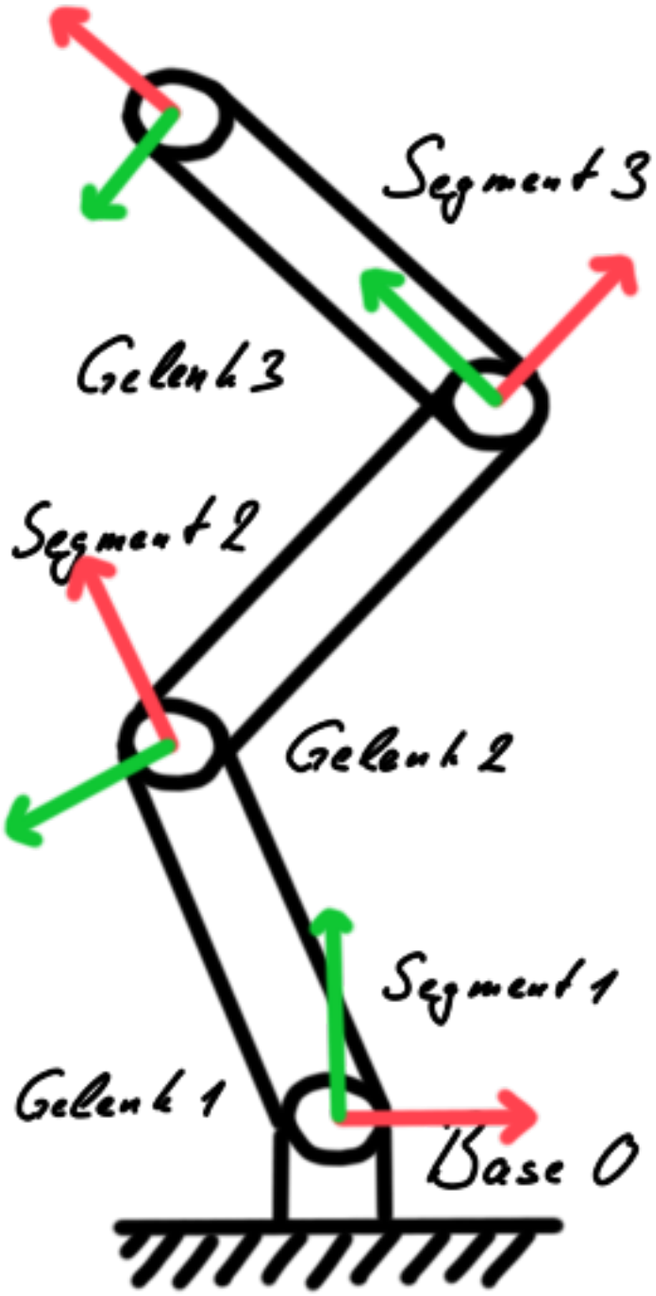
- Segmente erhalten Nummern von 1 bin *n*
- Die Basis erhält die Nummer 0
- Gelenk *n* verbindet Segment *n-1* mit Segment *n*

<u>Schritt 2:</u> Koordinatensysteme festlegen

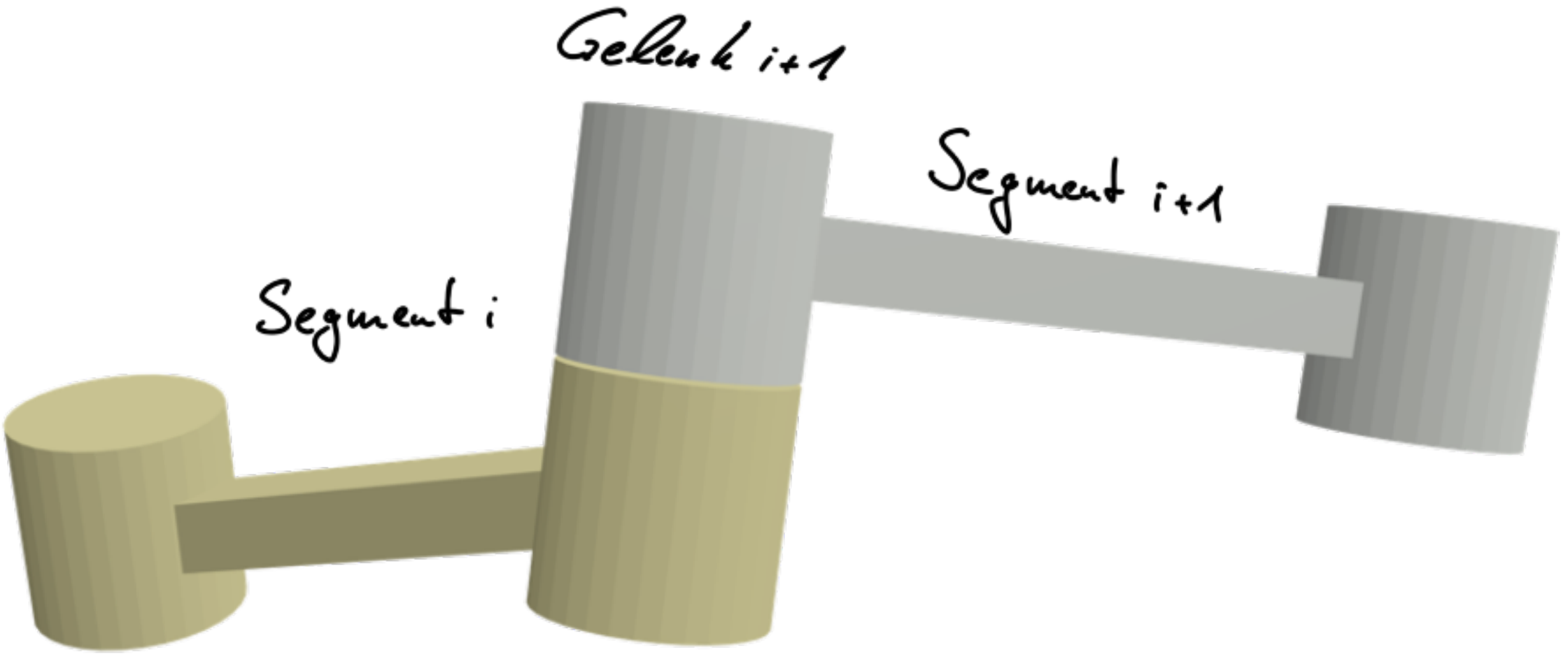
- Koordinatensystem der Basis in Gelenk 1 und parallel zum Weltkoordinatensystem
- Koordinatensystem des Segments *n* “nah” an Gelenkt *n+1*

<u>Schritt 3:</u> Gelenkachsen festlegen

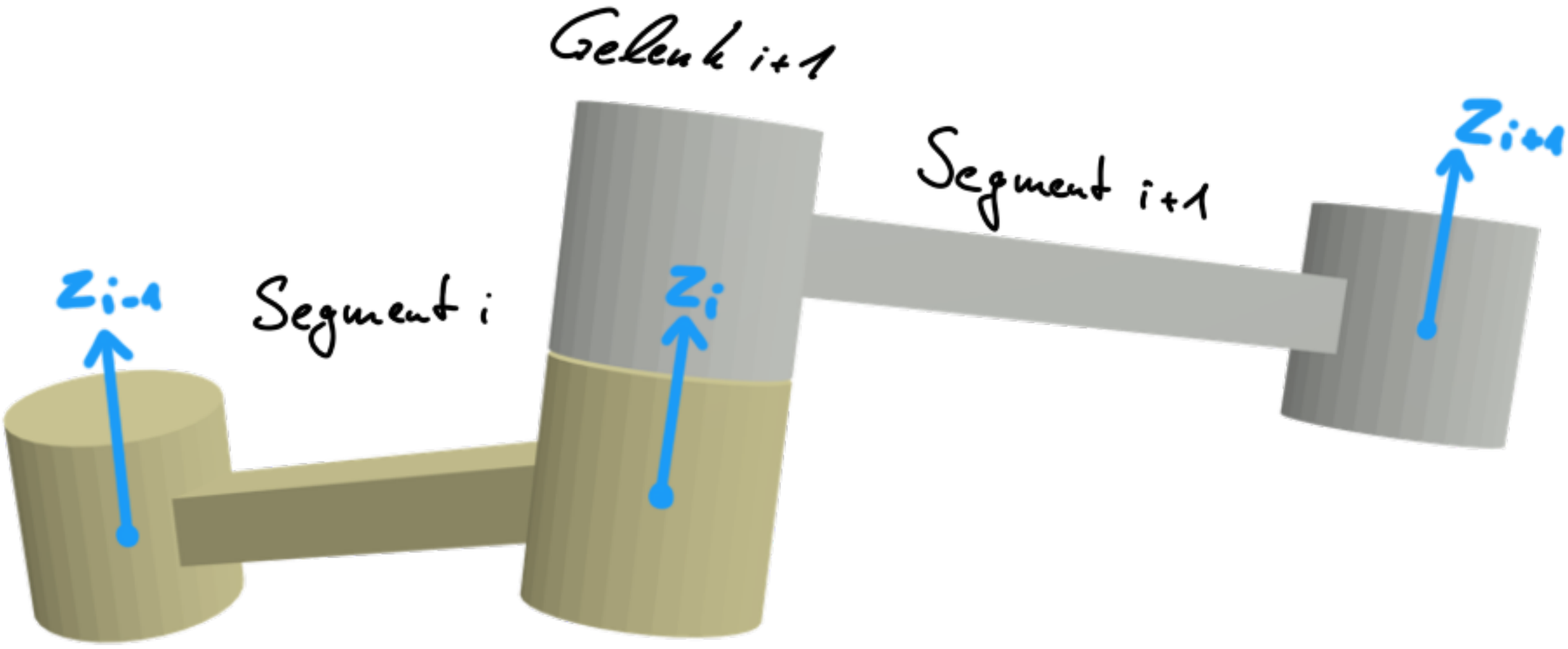
- Z-Achsen sind die Gelenkachsen
- X-Achsen verbinden die Z-Achsen (Kreuzprodukt)
- Y-Achsen folgen der Rechte-Hand-Regel



Denavit-Hartenberg — Nummerierung

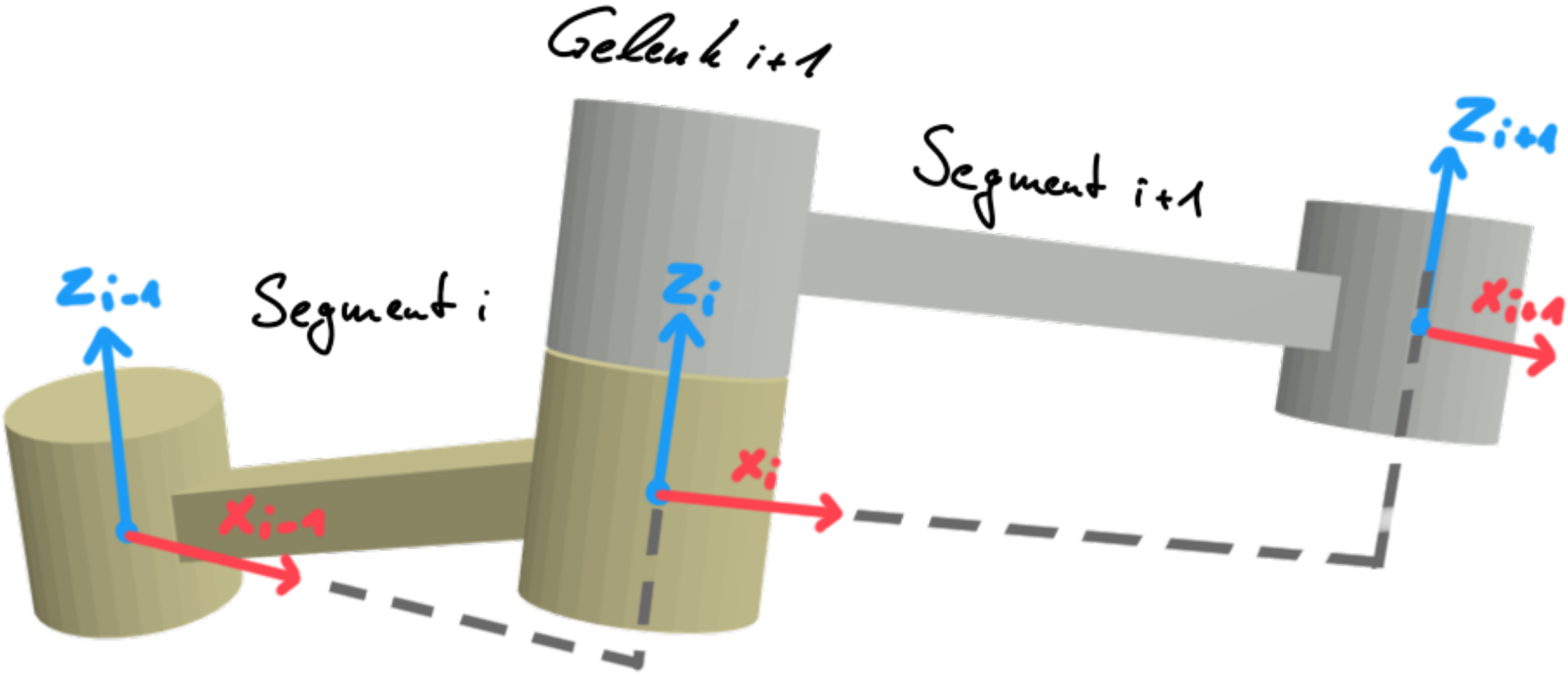


Denavit-Hartenberg — Z-Achsen festlegen



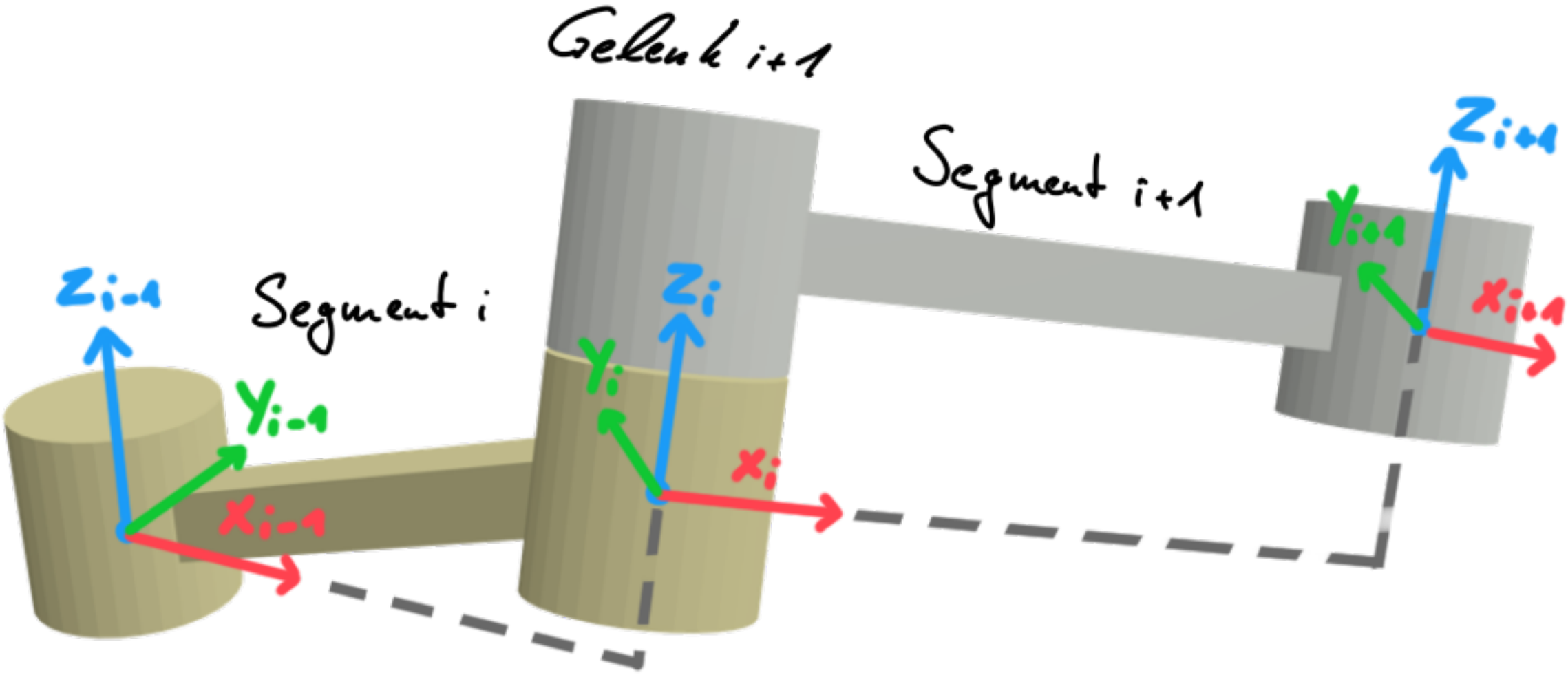
- Z-Achsen sind die Gelenkachsen des folgenden Gelenks

Denavit-Hartenberg — X-Achsen



- Z-Achsen sind die Gelenkachsen des folgenden Gelenks
- X-Achsen “verbinden” die Z-Achsen

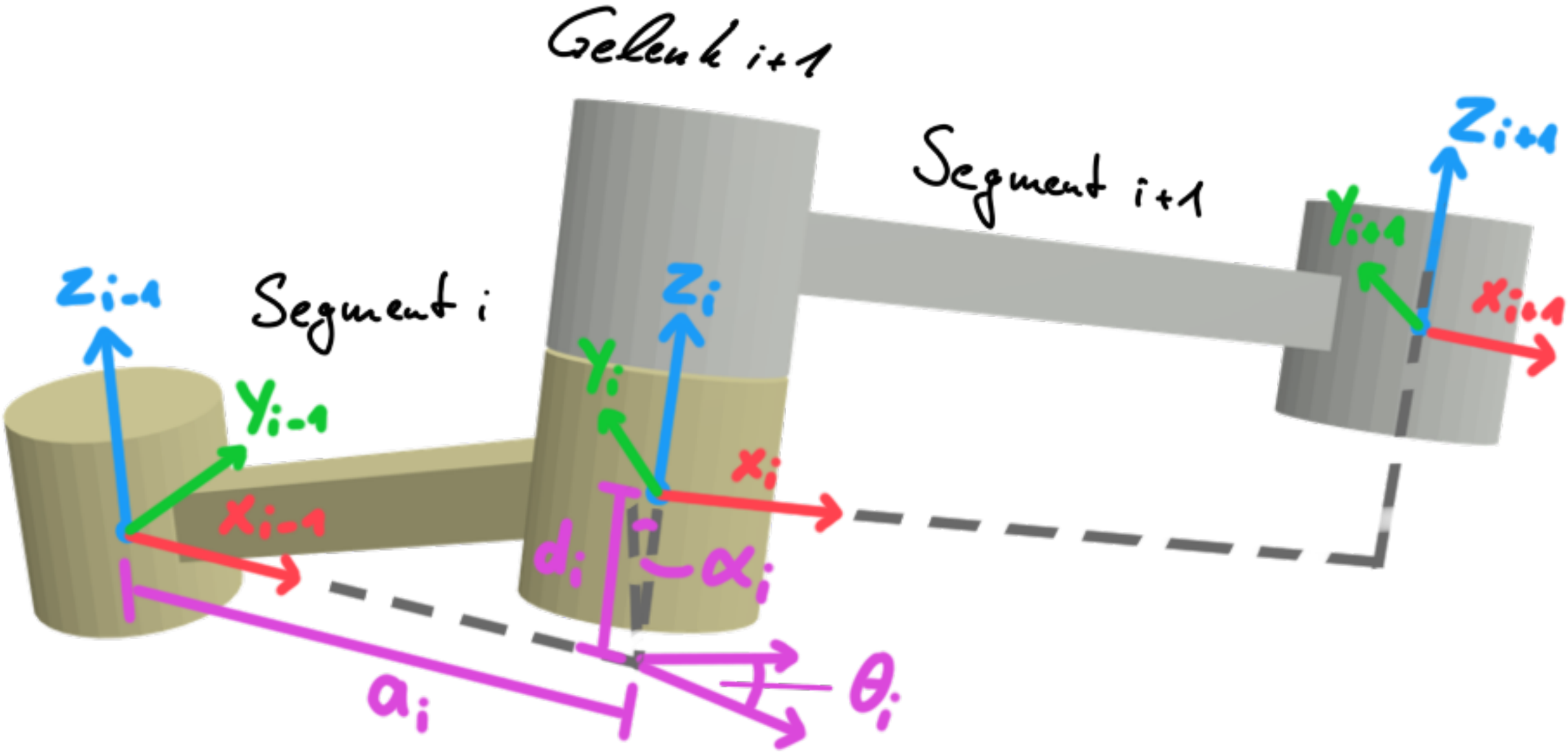
Denavit-Hartenberg — Y-Achsen



- Z-Achsen sind die Gelenkachsen des folgenden Gelenks
- X-Achsen “verbinden” die Z-Achsen

- Y-Achsen finden man durch die Rechte-Hand-Regel

Denavit-Hartenberg — Distanzen



- Z-Achsen sind die Gelenkachsen des folgenden Gelenks
- X-Achsen “verbinden” die Z-Achsen

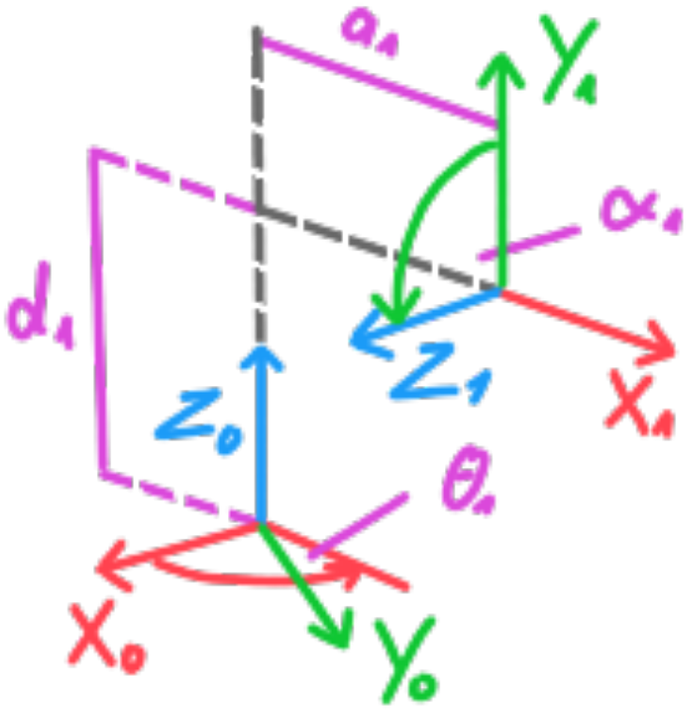
- Y-Achsen finden man durch die Rechte-Hand-Regel
- Gemeinsame Normale: $x_i = z_{i-1} \times z_i$
- Parameter festlegen

Im Buch werden unterschiedliche Notation verwendet

Denavit-Hartenberg — Schritt 4

- Die Transformation von Gelenk *i-1* zu Gelenk *i* kann nun bestimmt werden durch:

$$A_i(\theta_i, d_i, \alpha_i, a_i) = \mathrm{Rot}(z_i, \theta_i)\mathrm{Trans}(z_i, d_i)\mathrm{Trans}(x_i, a_i)\mathrm{Rot}(x_i, \alpha_i)$$



$$= \begin{pmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\cdot \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

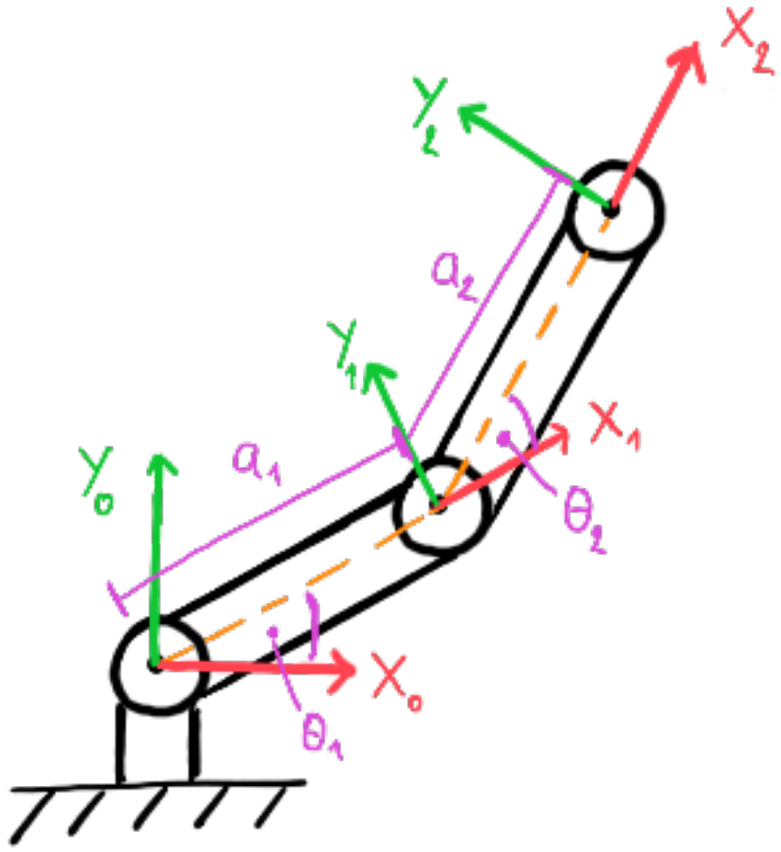
$$= \begin{pmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Denavit-Hartenberg — Beispiel

Segment	a_i	α_i	d_i	θ_i
1	a_1	0	0	θ_1
2	a_2	0	0	θ_2

$$A_1 = \begin{pmatrix} c_1 & -s_1 & 0 & a_1 c_1 \\ s_1 & c_1 & 0 & a_1 s_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

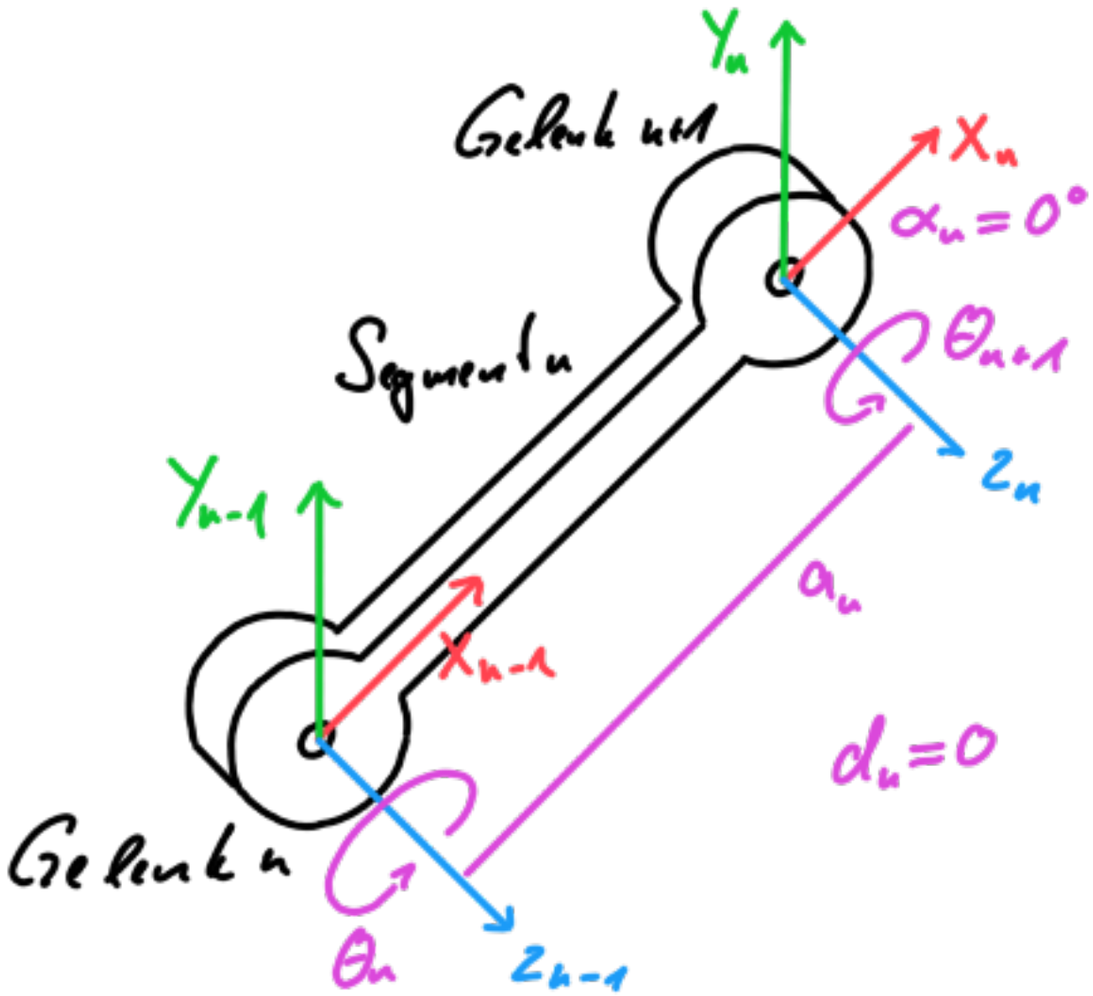
$$A_2 = \begin{pmatrix} c_2 & -s_2 & 0 & a_2 c_2 \\ s_2 & c_2 & 0 & a_2 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

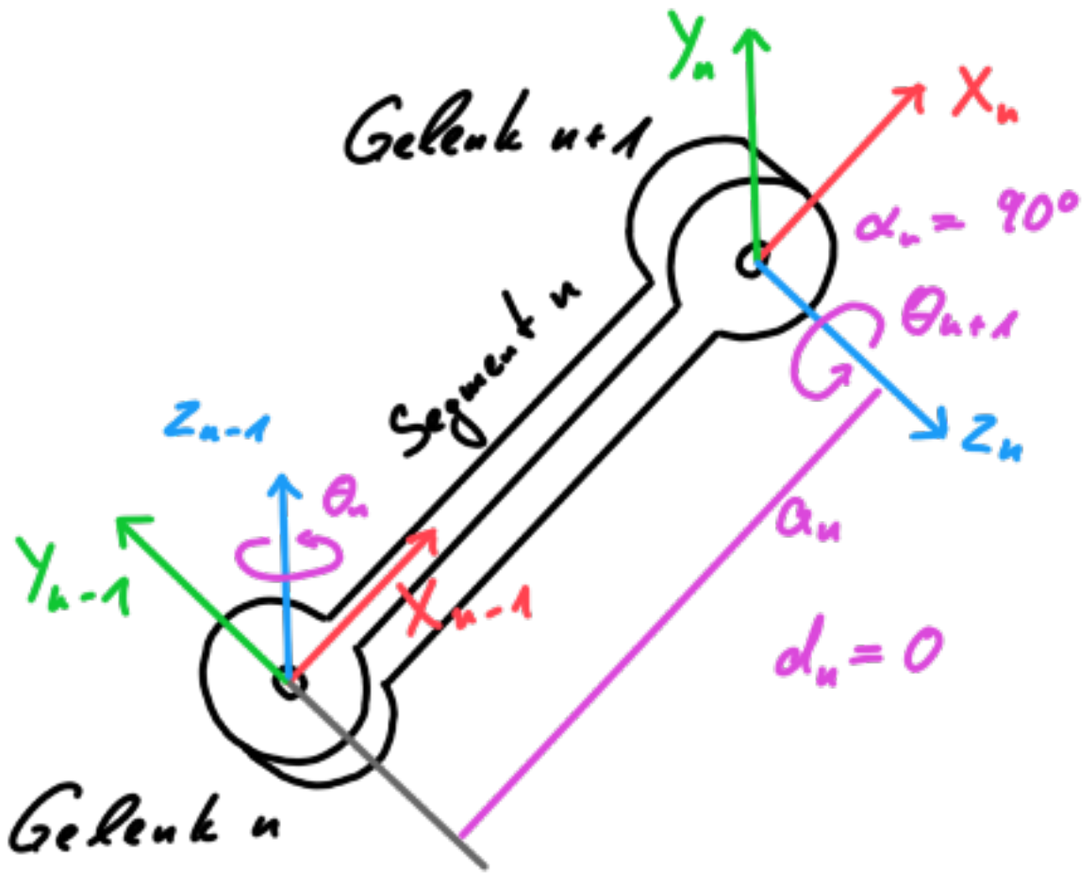


$${}^0T_1 = A_1$$

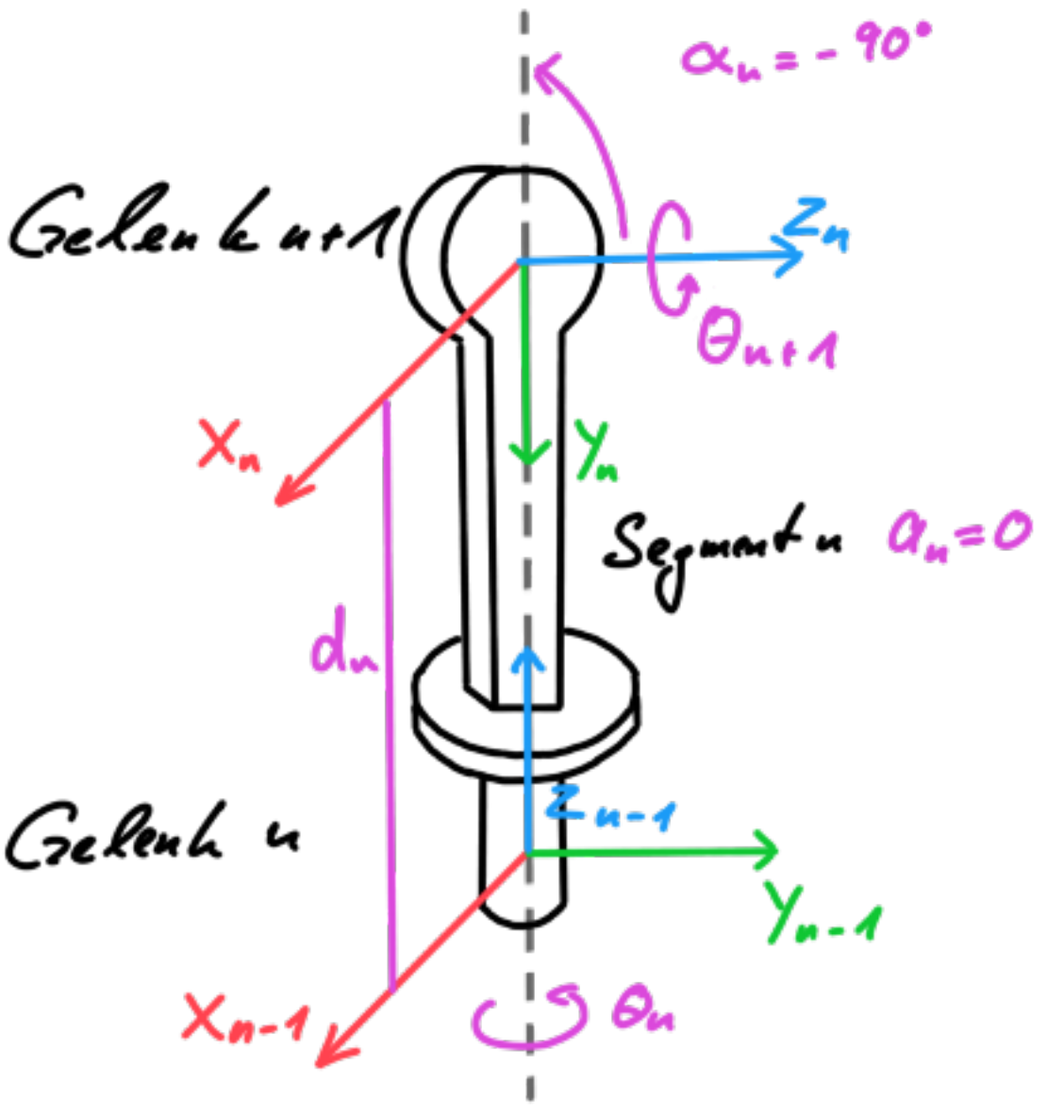
$${}^0T_2 = A_1 A_2$$

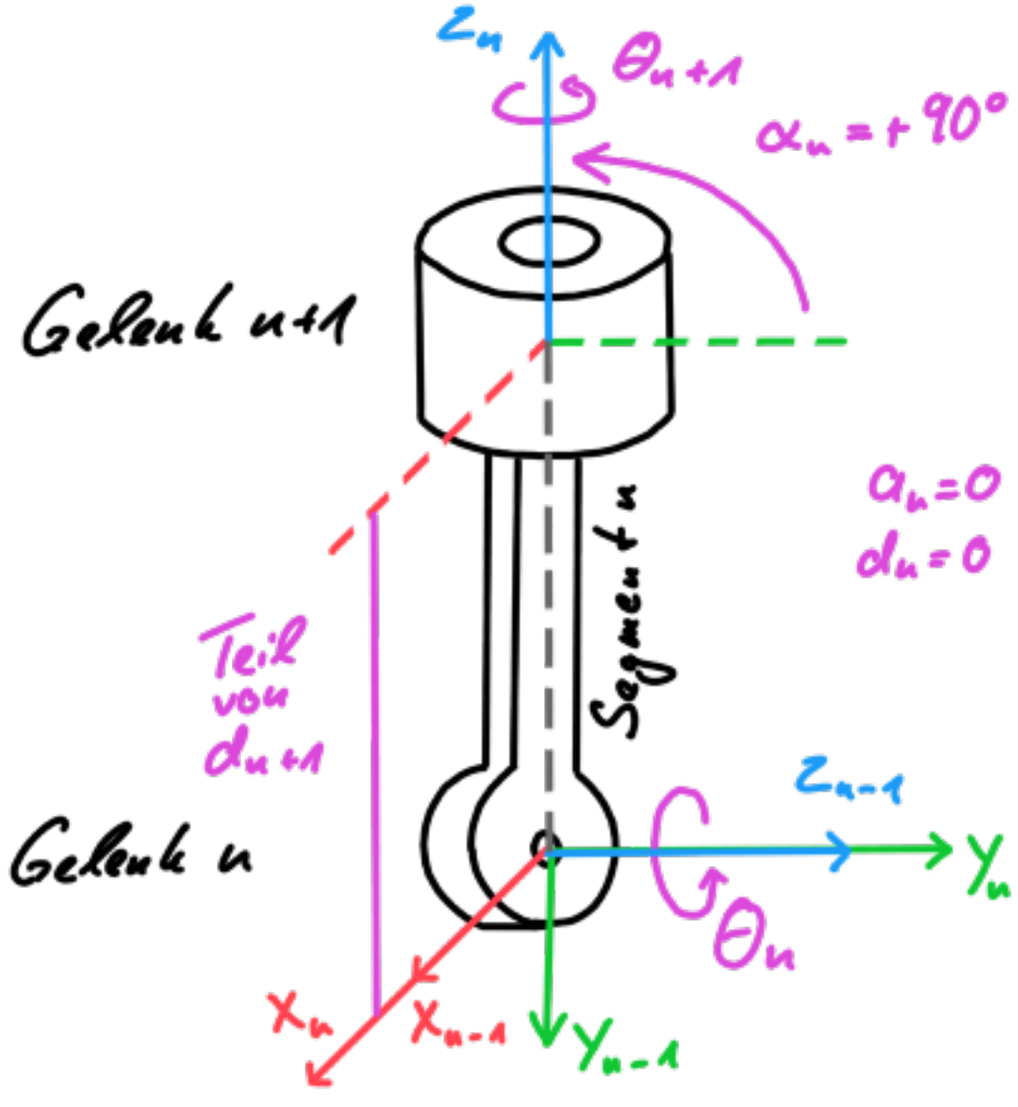
$$= \begin{pmatrix} c_{12} - s_{12} & -c_2 s_1 - c_1 s_2 & 0 & a_1 c_1 + a_2 c_{12} - a_2 s_{12} \\ c_2 s_1 + c_1 s_2 & c_{12} - s_{12} & 0 & a_1 s_1 + a_2 c_2 s_1 + a_2 c_1 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



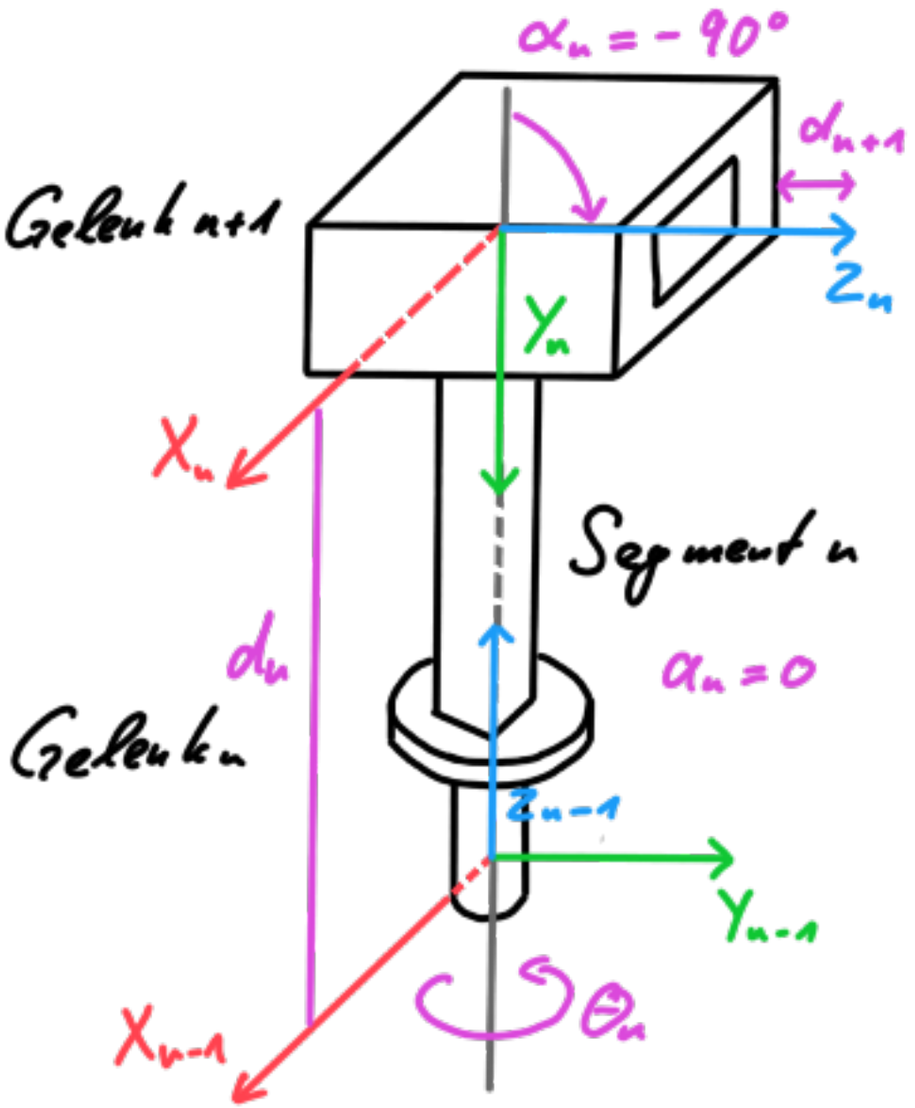


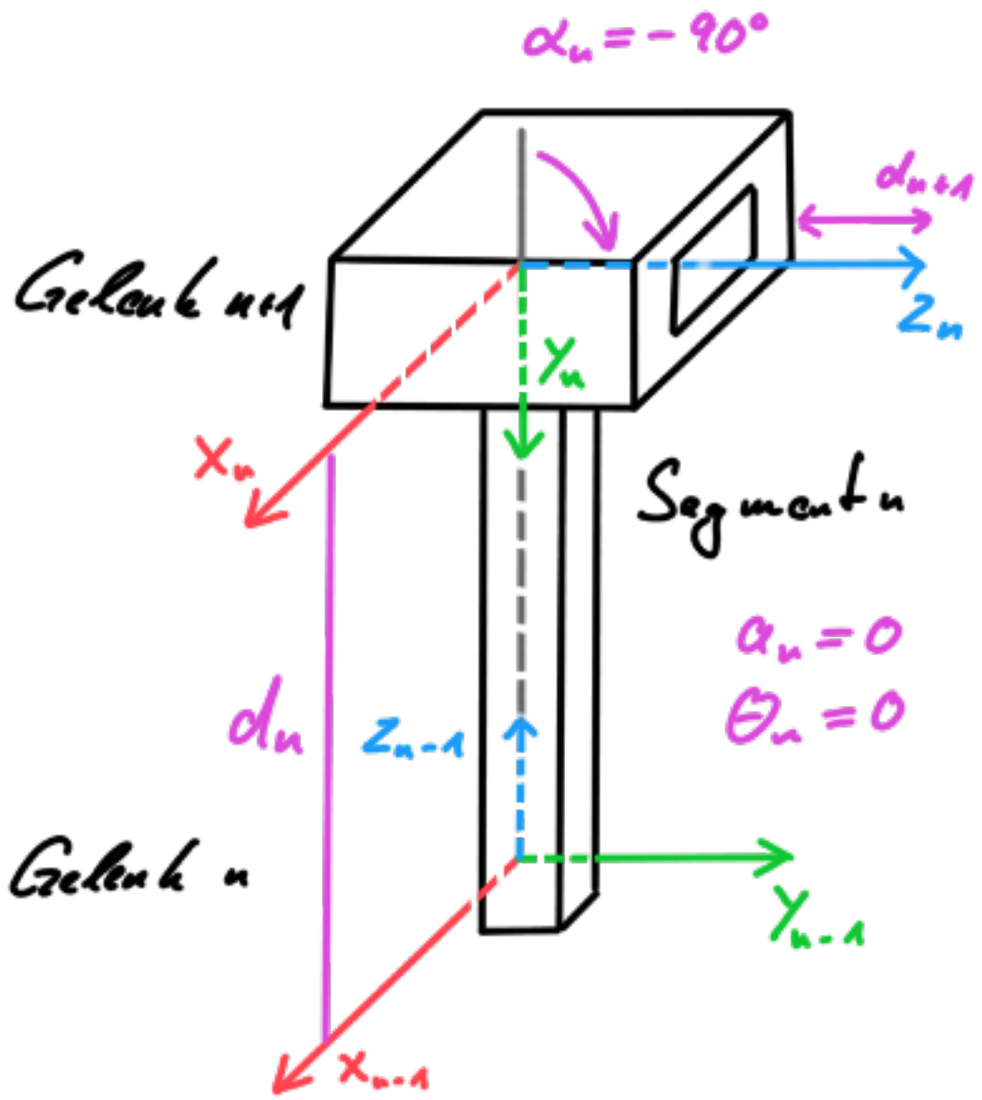
a — Segmentlänge
α — Segmentwinkel
d — Gelenkoffset
θ — Gelenkwinkel





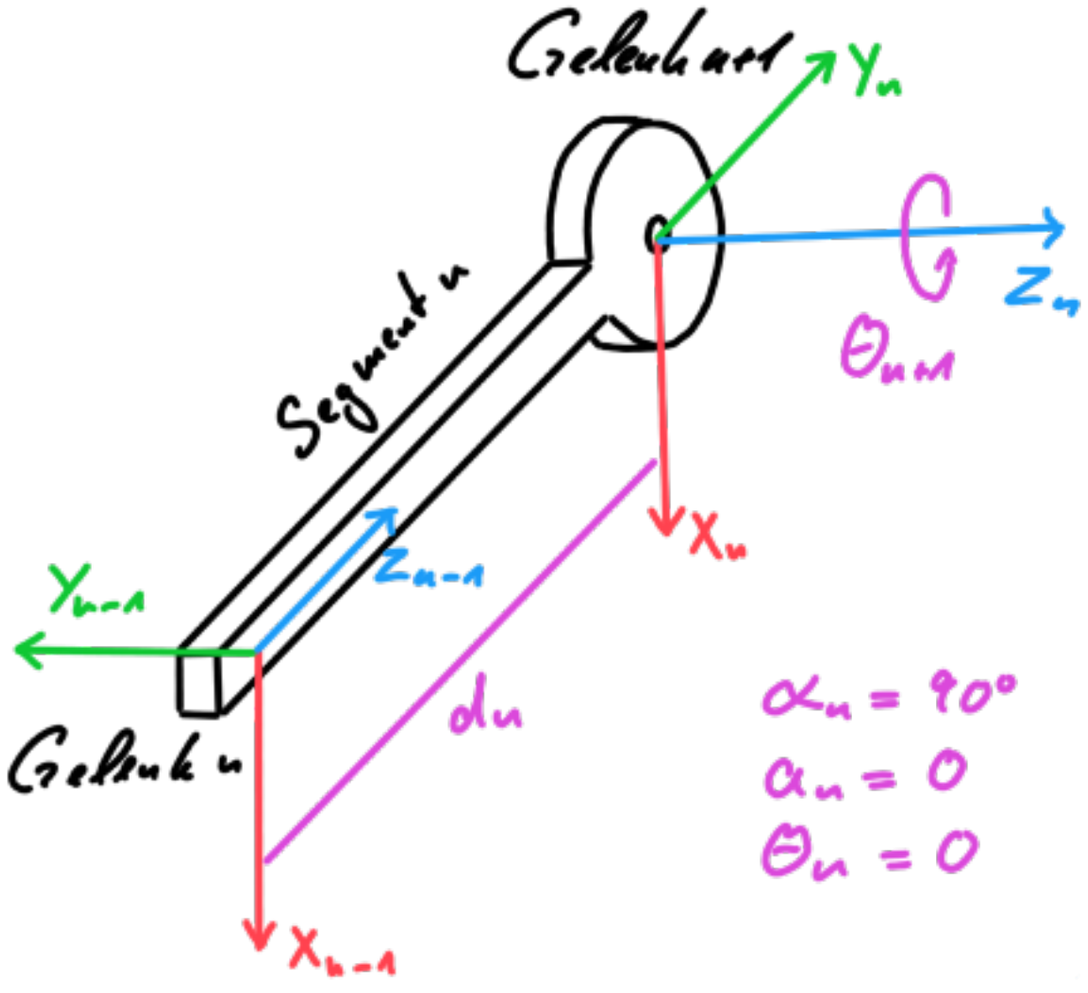
a — Segmentlänge
α — Segmentwinkel
d — Gelenkoffset
θ — Gelenkwinkel

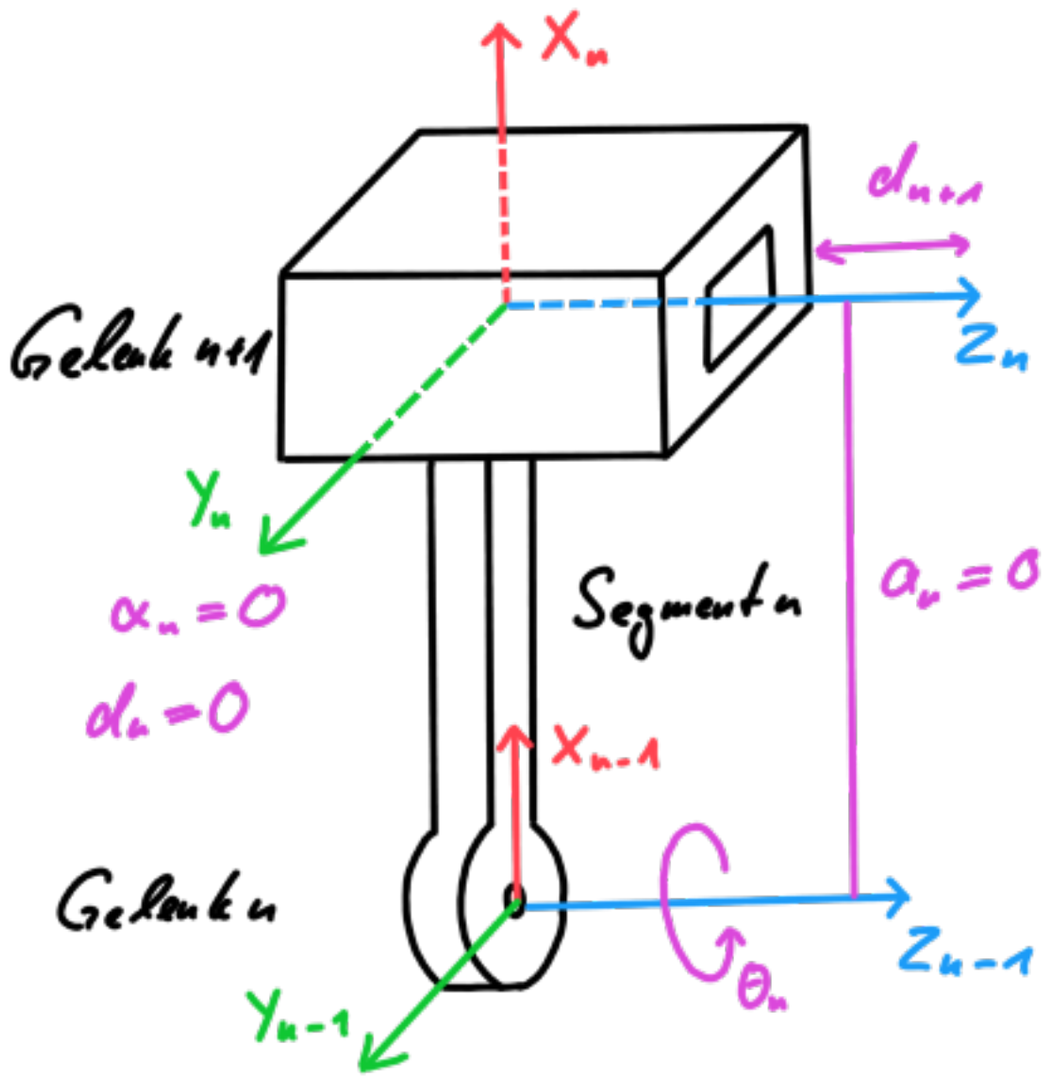




a — Segmentlänge
α — Segmentwinkel

d — Gelenkoffset
θ — Gelenkwinkel



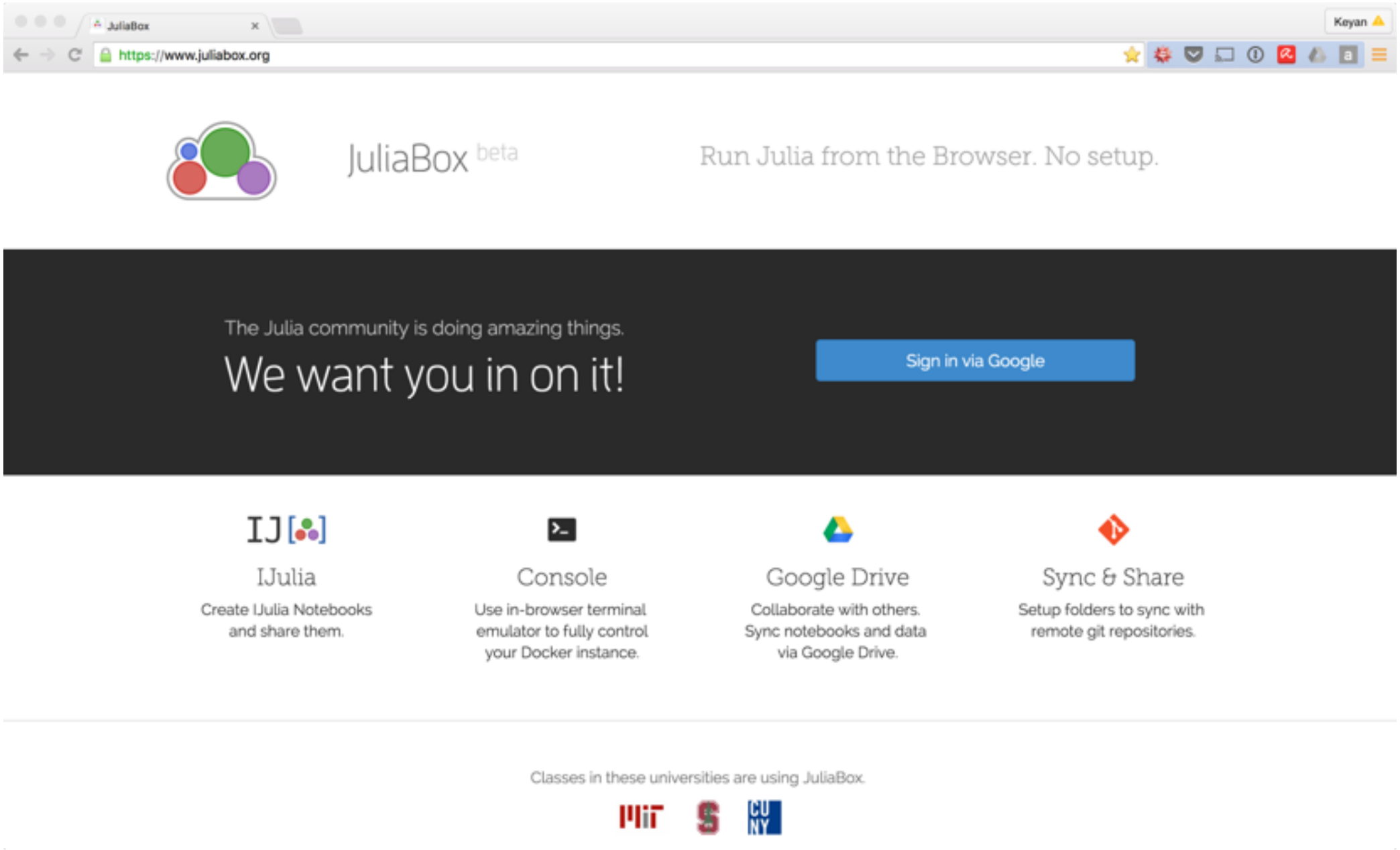


a — Segmentlänge
α — Segmentwinkel
d — Gelenkoffset
θ — Gelenkwinkel

Julia Code Beispiele — www.julialang.org

- Freie Software für numerisches Rechnen
- Grosse, sehr aktive Community
- MATLAB-ähnlicher Syntax, fast C++ Geschwindigkeit (Faktor 2)
- Learn Julia in Y Minutes: https://learnxinyminutes.com/docs/julia/
- GIT Repository für die Vorlesung: https://github.com/kzahedi/Robotik
- JuliaBox https://www.juliabox.org

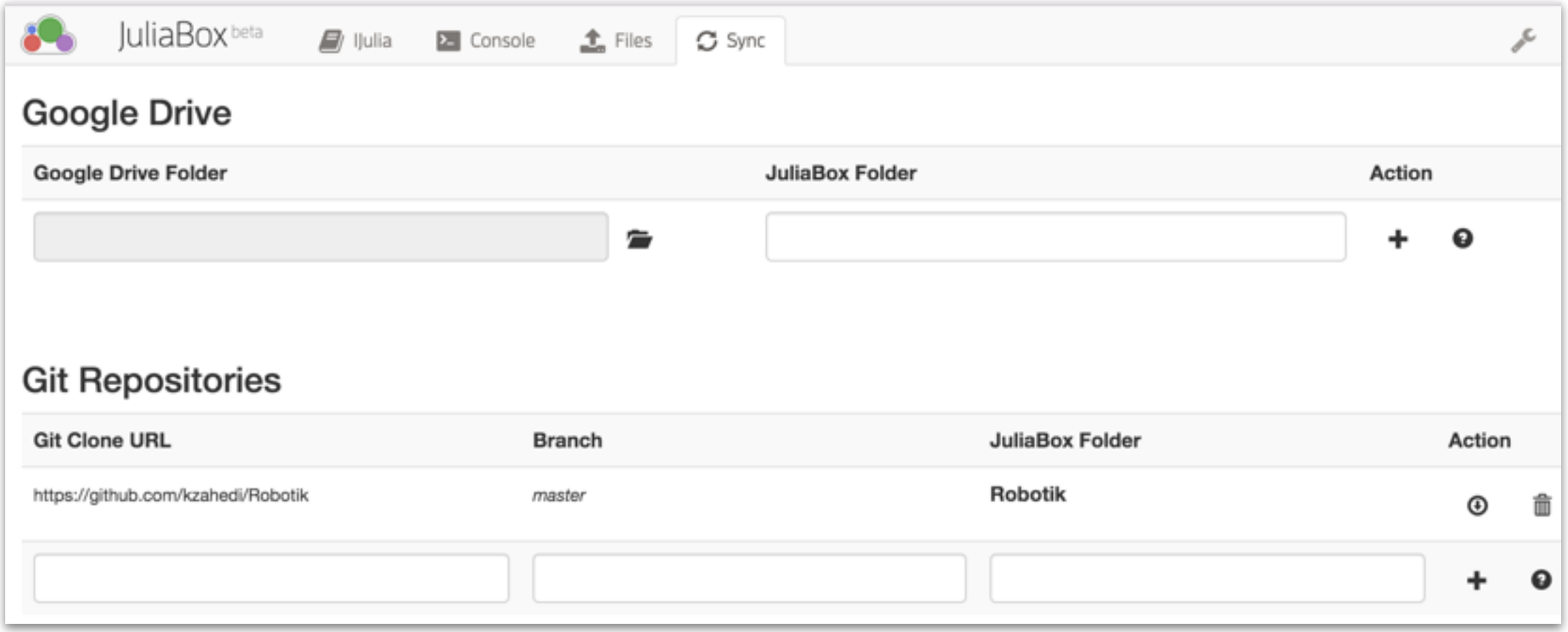
Julia Box — https://www.juliabox.org



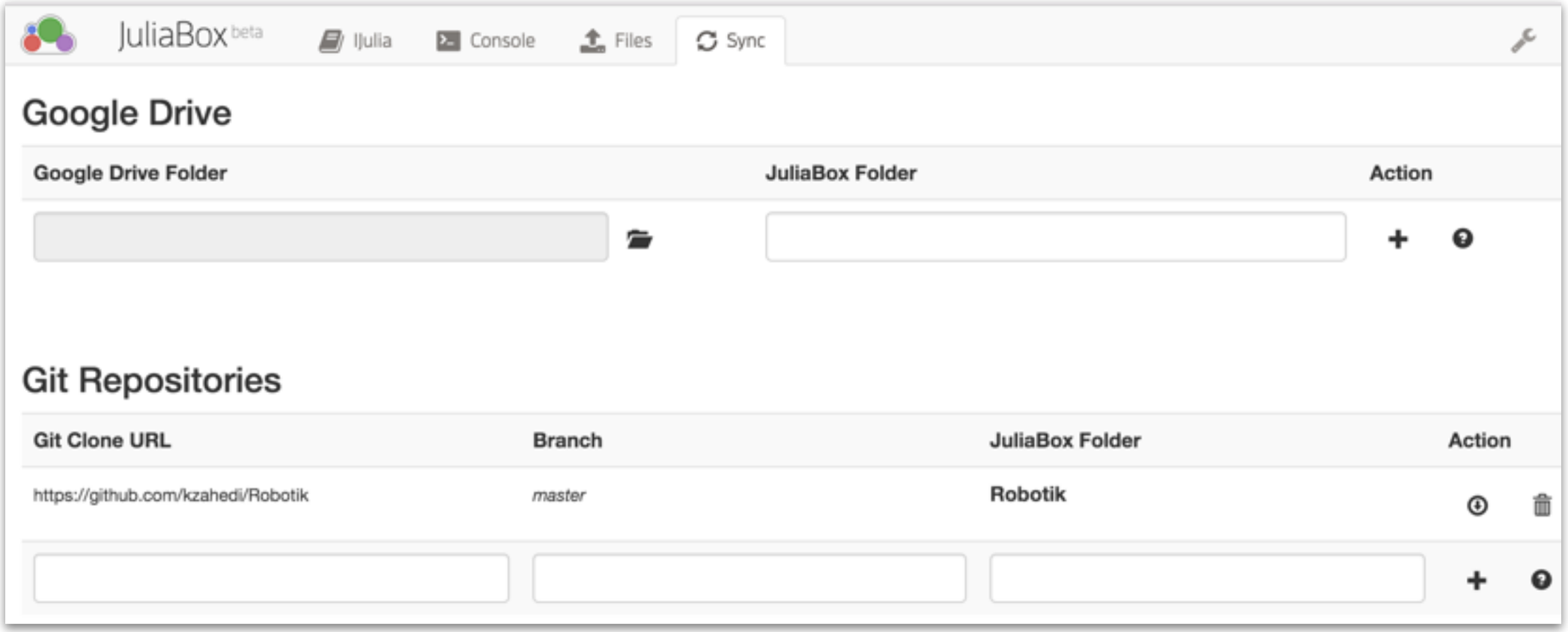
- 8 CPUs
- 6 GB RAM
- 500 MB Storage
- Sessions: 4 Stunden

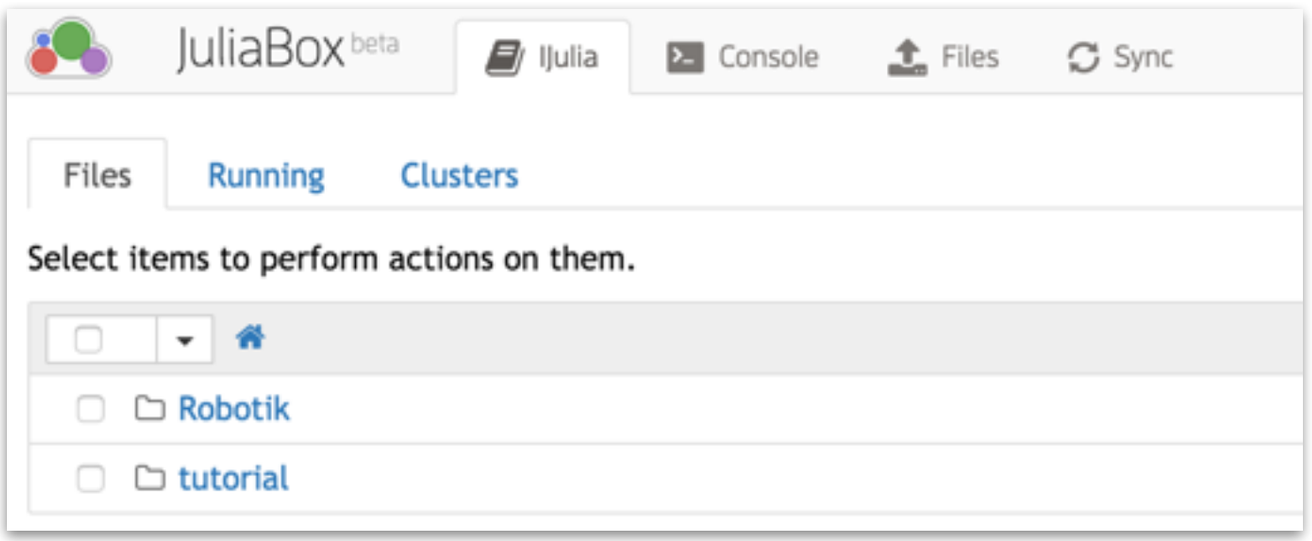
- Synchronisation über Google Drive & Github
- Vollwertige Konsole (Linux)

JuliaBox — https://www.juliabox.org

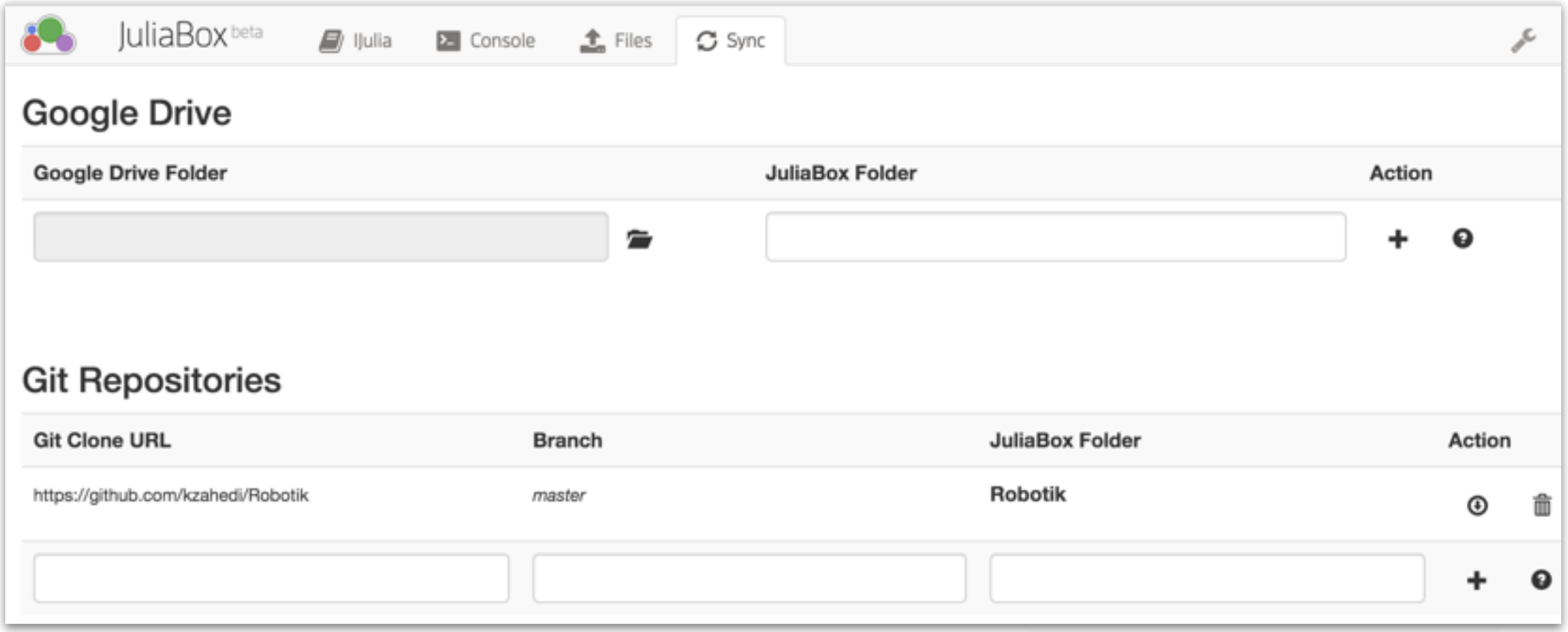


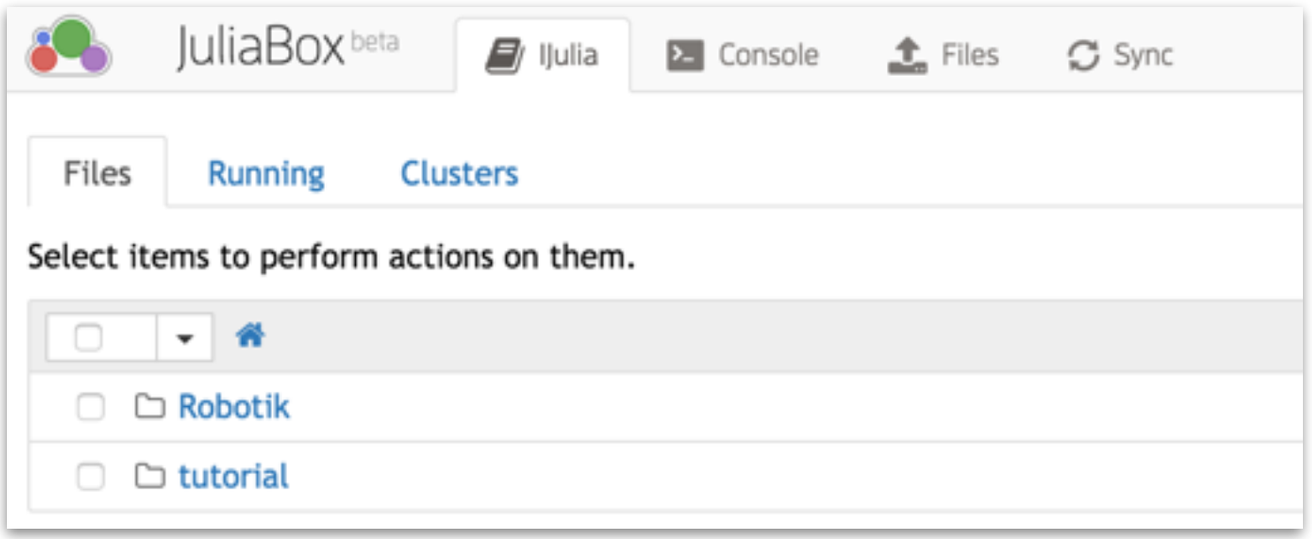
JuliaBox — https://www.juliabox.org

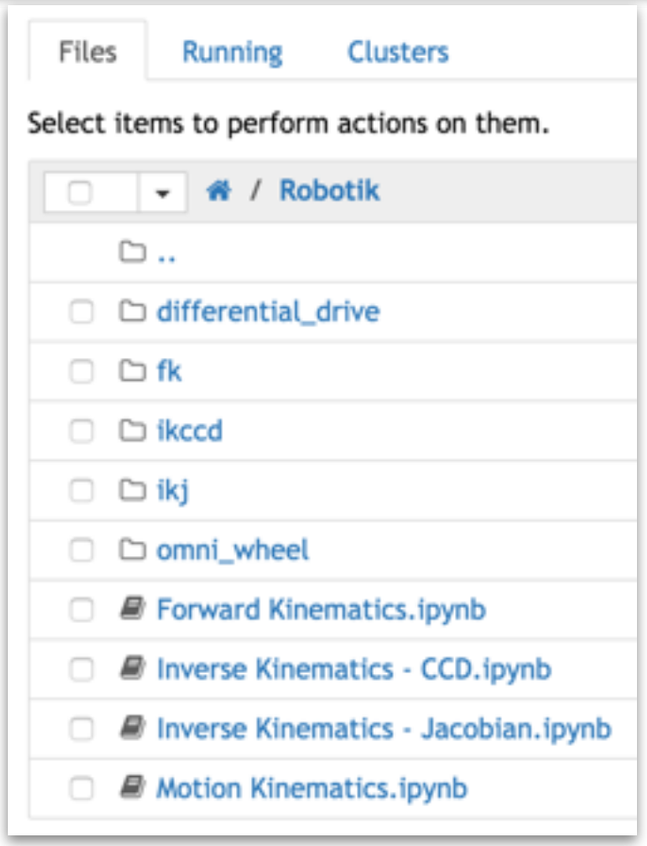




JuliaBox — https://www.juliabox.org







Direkte Kinematik / Vorwärtskinematik

```
#Pkg.add("PyPlot")
#Pkg.build("PyCall")
using PyPlot
```

Homogene Transformationsmatrizen

```
RZ(θ) = [
    [cos(θ) -sin(θ) 0 0];
    [sin(θ) cos(θ) 0 0];
    [0 0 1 0];
    [0 0 0 1]
]

RX(α) = [
    [1 0 0 0]
    [0 cos(α) -sin(α) 0];
    [0 sin(α)  cos(α) 0];
    [0 0 0 1]
]

TZ(d) = [
    [1 0 0 0];
    [0 1 0 0];
    [0 0 1 d];
    [0 0 0 1]
]

TX(a) = [
    [1 0 0 a];
    [0 1 0 0];
    [0 0 1 0];
    [0 0 0 1]
]

A(θ, d, α, a) = RZ(θ) * TZ(d) * TX(a) * RX(α)
```

Definition Roboter

```
A1(θ) = A(θ,0,0,1)
A2(θ) = A(θ,0,0,0.75)
A3(θ) = A(θ,0,0,0.5)
```

Vorwärtskinematik

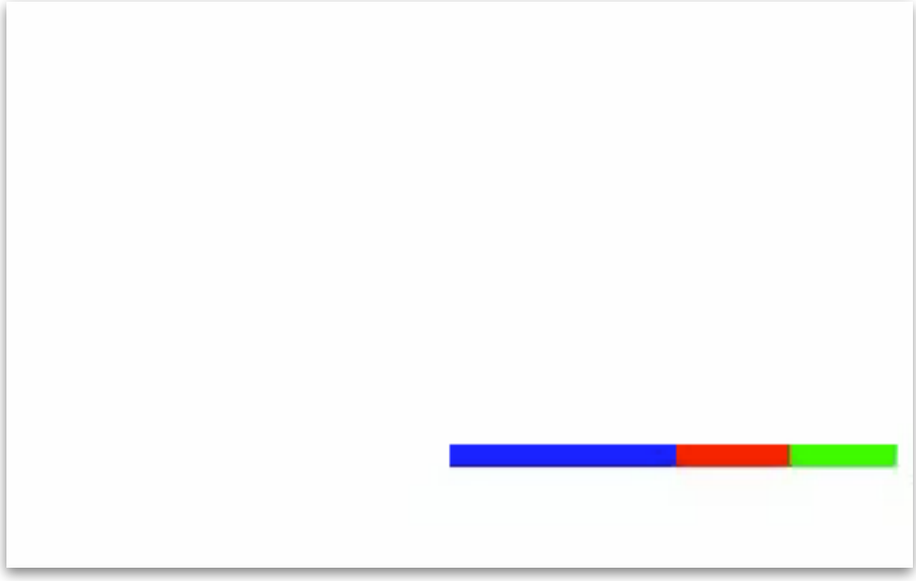
```
function ForwardKinematics(θ1, θ2, θ3)
    base = [0;0;0;1]

    p1 = A1(θ1) * base
    p2 = A1(θ1) * A2(θ2) * base
    p3 = A1(θ1) * A2(θ2) * A3(θ3) * base

    [p1 p2 p3]
end
```

Direkte Kinematik / Vorwärtskinematik

Plot Funktionen

```
function PlotRoboter(θ1, θ2, θ3)
    coords = ForwardKinematics(θ1, θ2, θ3)

    fig = figure(figsize=(8,5))
    xlim(-4,4)
    ylim(-1,4)
    xticks([])
    yticks([])

    plot([0,           coords[1,1]], [0,           coords[2,1]], color="red",   lw = 5.0)
    plot([coords[1,1], coords[1,2]], [coords[2,1], coords[2,2]], color="green", lw = 5.0)
    plot([coords[1,2], coords[1,3]], [coords[2,2], coords[2,3]], color="blue",  lw = 5.0)
    fig

end
```

```
ioff()

θ1 = collect(0:0.01:π)
θ2 = collect(0:0.02:2π)
θ3 = collect(0:0.04:4π)

for i = 1:length(θ1)
    fig = PlotRoboter(θ1[i], θ2[i], θ3[i]);
    savefig("fk/robot_$(i-1).png", bbox_inches="tight")
    close(fig)
end

ion()
```



Direkte Kinematik (Vorwärtskinematik) & Inverse Kinematik

- Direkte Kinematik (Forward kinematics)
  - Gegeben: Parameter θ aller Gelenke
  - Ziel: Bestimmen der Pose des Endeffektors
    - Siehe vorheriges Beispiel
- Inverse Kinematik (Inverse kinematics)
  - Gegeben: Position des Endeffektors
  - Ziel: Bestimmen der Parameter aller Gelenke

Inverse Kinematik

- <u>Problemstellung:</u> Wir wollen mit dem Endeffektor eine vorgegebene Position erreichen und benötigen die Gelenkparameter θ_i für jedes Gelenk

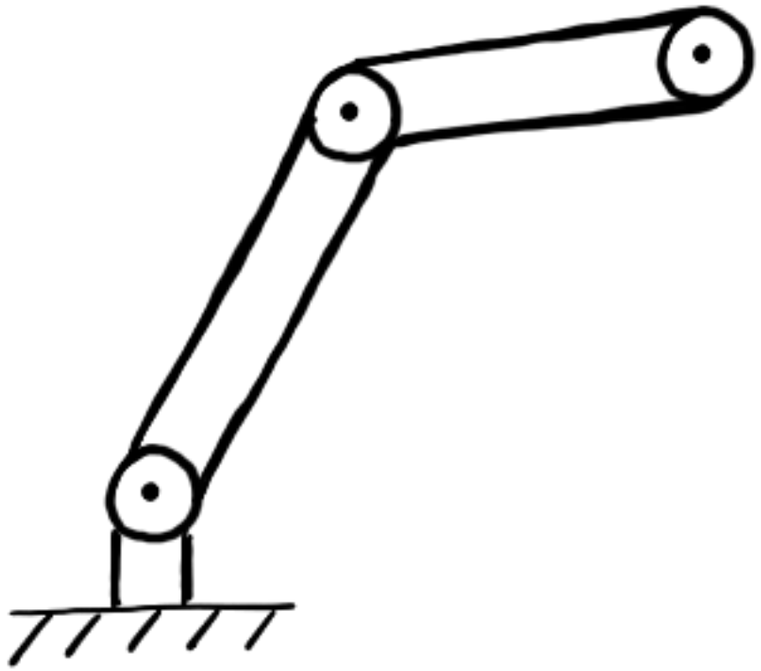
- <u>Lösungswege:</u>

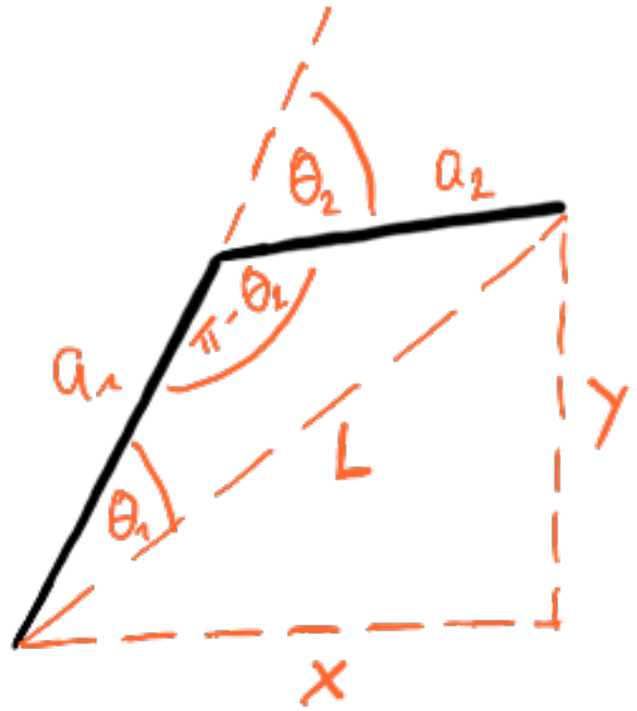
  1. Analytische Verfahren (closed-form solutions)

  2. Numerische Verfahren

- <u>Definition:</u> Arbeitsbereich (Workspace)
  Alle Positionen, die der Endeffektor erreichen kann

Beispiel für eine analytisches Verfahren





2 Probleme:

1. Die Lösung ist spezifisch für einen Roboter
2. Nicht praktikabel für komplexe Systeme

$$L = \sqrt{x^2 + y^2}$$

$$L_1 - L_2 \leq L \leq L_1 + L_2$$

$$\cos(\theta) = \frac{x}{L}$$

$$\cos(\theta_1 - \theta) = \frac{a_2^2 - L^2 + a_2^2}{2a_1 L}$$

$$\cos(\pi - \theta_2) = \frac{L^2 - a_1^2 - a_2^2}{2a_1 a_2}$$

Inverse Kinematik — Jacobian

e – Position des Endeffektors

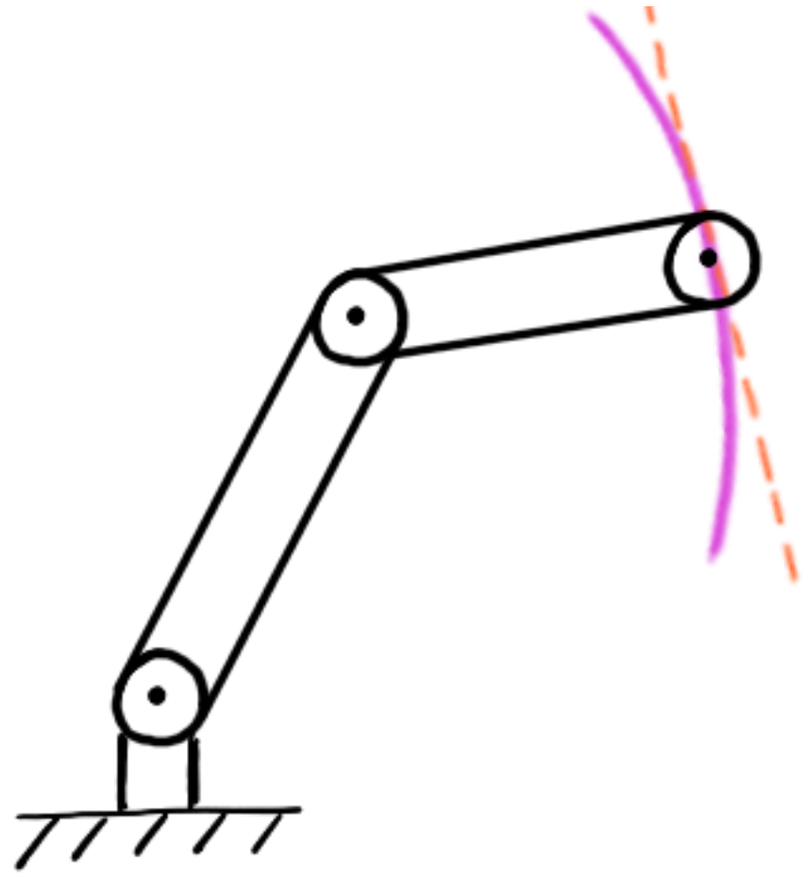
θ – Alle Gelenkparameter

$e = f(\theta)$ – Vorwärtskinematik

$\theta = f^{-1}(e)$ – Inverse Kinematik

Jacobi-Matrix (Ableitung in einem Punkt)

$$J = \frac{\Delta e}{\Delta \theta} = \begin{pmatrix} \frac{\Delta e_x}{\Delta \theta_1} & \frac{\Delta e_x}{\Delta \theta_2} & \cdots & \frac{\Delta e_x}{\Delta \theta_m} \\ \frac{\Delta e_y}{\Delta \theta_1} & \frac{\Delta e_y}{\Delta \theta_2} & \cdots & \frac{\Delta e_y}{\Delta \theta_m} \\ \frac{\Delta e_z}{\Delta \theta_1} & \frac{\Delta e_z}{\Delta \theta_2} & \cdots & \frac{\Delta e_z}{\Delta \theta_m} \end{pmatrix}$$



Änderung der Parameter um die Position
in die gewünschte Richtung zu ändern

$\Delta\theta = J^{-1}\Delta e$

$\Delta\theta = (J^T J)^{-1} J^T \Delta e$ [Moore-Penrose pseudo inverse]

2 Probleme:

1. Nicht immer lösbar
2. Sehr teuer zu berechnen

Inverse Kinematik — Jacobian

e – Position des Endeffektors

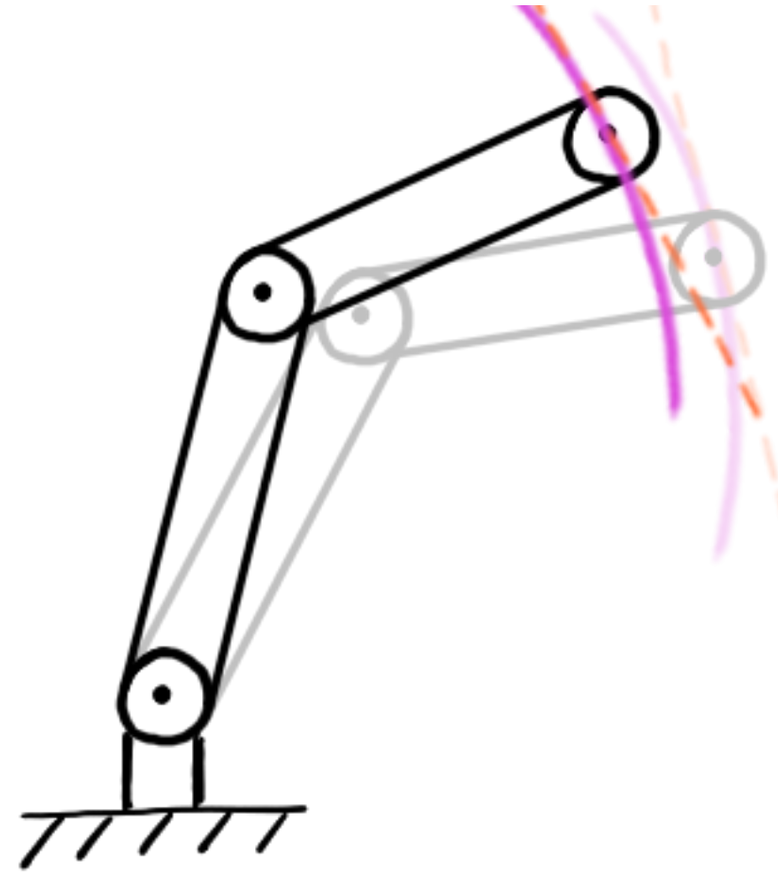
θ – Alle Gelenkparameter

$e = f(\theta)$ – Vorwärtskinematik

$\theta = f^{-1}(e)$ – Inverse Kinematik

Jacobi-Matrix (Ableitung in einem Punkt)

$$J = \frac{\Delta e}{\Delta \theta} = \begin{pmatrix} \frac{\Delta e_x}{\Delta \theta_1} & \frac{\Delta e_x}{\Delta \theta_2} & \cdots & \frac{\Delta e_x}{\Delta \theta_m} \\ \frac{\Delta e_y}{\Delta \theta_1} & \frac{\Delta e_y}{\Delta \theta_2} & \cdots & \frac{\Delta e_y}{\Delta \theta_m} \\ \frac{\Delta e_z}{\Delta \theta_1} & \frac{\Delta e_z}{\Delta \theta_2} & \cdots & \frac{\Delta e_z}{\Delta \theta_m} \end{pmatrix}$$



Änderung der Parameter um die Position
in die gewünschte Richtung zu ändern

$\Delta\theta = J^{-1}\Delta e$

$\Delta\theta = (J^T J)^{-1} J^T \Delta e$ [Moore-Penrose pseudo inverse]

2 Probleme:

1. Nicht immer lösbar
2. Sehr teuer zu berechnen

Inverse Kinematik — Jacobi-Matrix

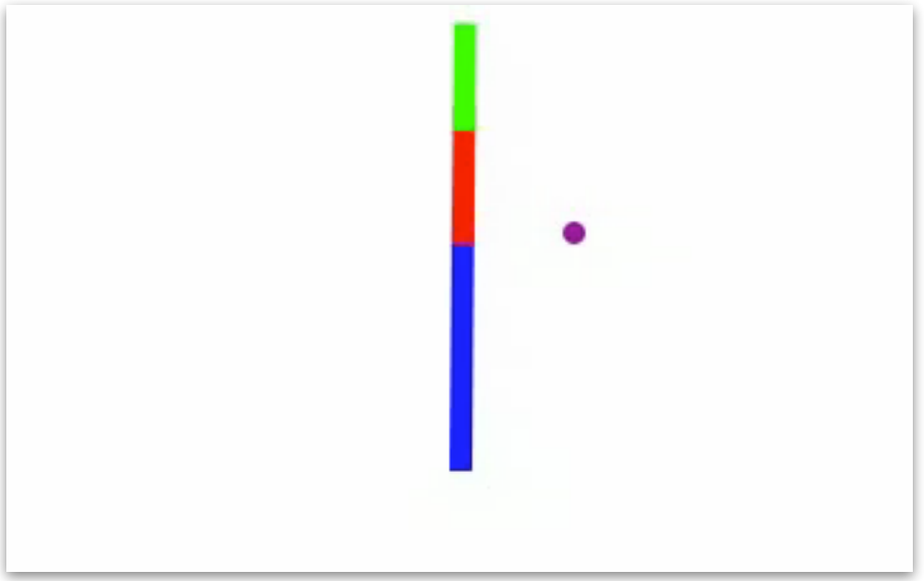
Definition Roboter

```
A1(θ) = A(θ,0,0,1)
A2(θ) = A(θ,0,0,0.75)
A3(θ) = A(θ,0,0,0.5)

function J(θ1, θ2, θ3)
    a = -2 * sin(θ1) - sin(θ1 + θ2) - 0.75 * sin(θ1 + θ2 + θ3)
    b = -1 * sin(θ1 + θ2) - 0.75 * sin(θ1 + θ2 + θ3)
    c = -0.75 * sin(θ1 + θ2 + θ3)

    d = 2 * cos(θ1) + cos(θ1 + θ2) + 0.75 * cos(θ1 + θ2 + θ3)
    e = 1 * cos(θ1 + θ2) + 0.75 * cos(θ1 + θ2 + θ3)
    f = 0.75 * cos(θ1 + θ2 + θ3)

    [ [a b c]; [d e f]]
end

Jinv(θ1, θ2, θ3) = pinv(J(θ1, θ2, θ3))

normalize(x) = x/norm(x)
```

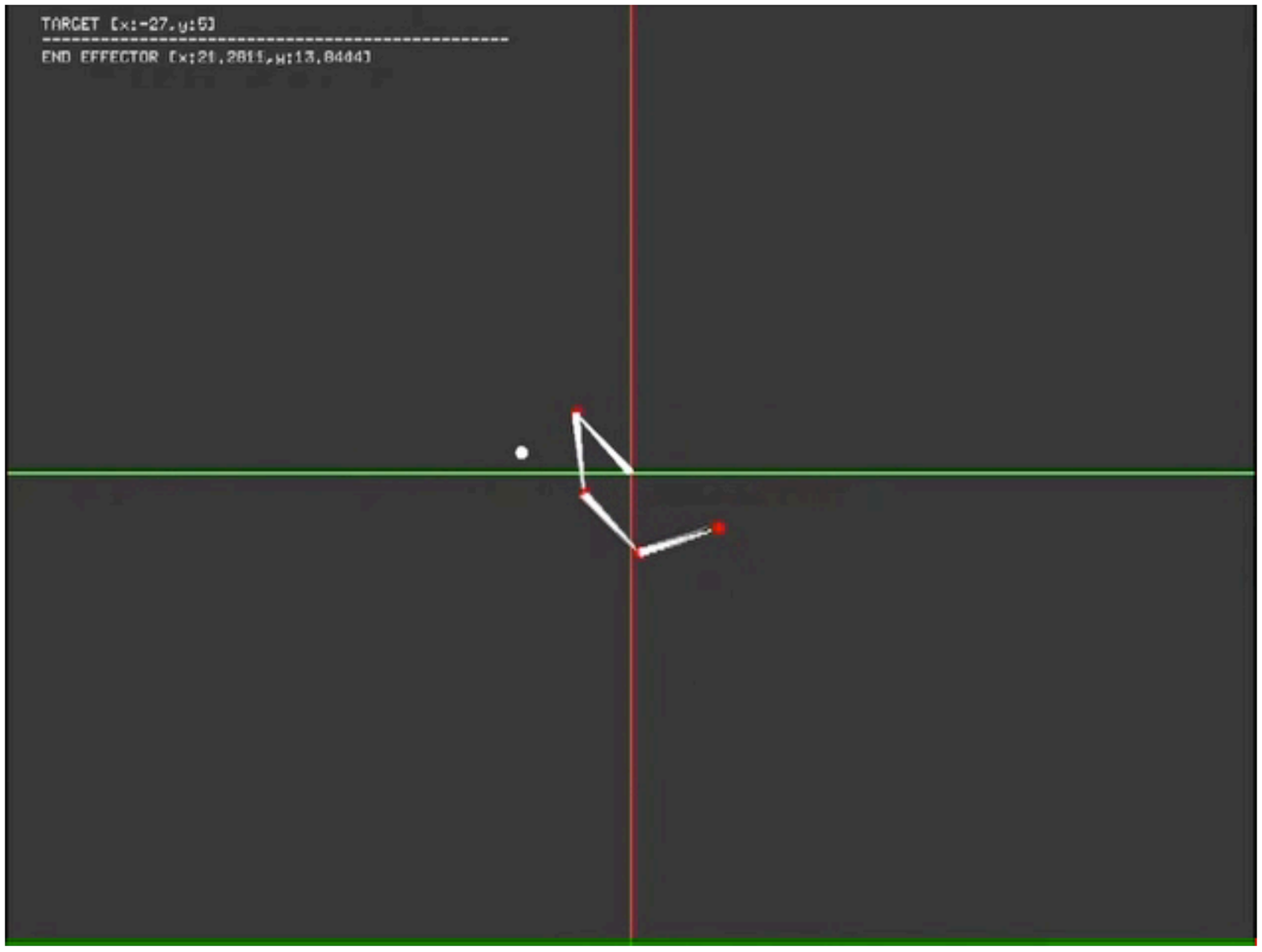
Inverse Kinematik

```
function InverseKinematics(goal, q)
    base = [0;0;0;1]
    δ = 0.01
    e = A1(q[1]) * A2(q[2]) * A3(q[3]) * base
    j = Jinv(q[1], q[2], q[3])
    d = reshape(goal - e[1:2], 2,1)
    step = j * d
    dist = sqrt(step[1] * step[1] + step[2] * step[2])
    if dist > δ
        step = normalize(step) * δ
    end
    q = q + step'
    q
end
```

```
q = [π/2 0.01 0.01]
goal = [1.0; 0.5]
j = 0

ioff()
for i = 1:250
    j = j + 1
    q = InverseKinematics(goal, q)
    fig = PlotRoboter(q,goal)
    savefig("ikj/robot_$(j-1).png", bbox_inches="tight")
    close(fig)
end
```



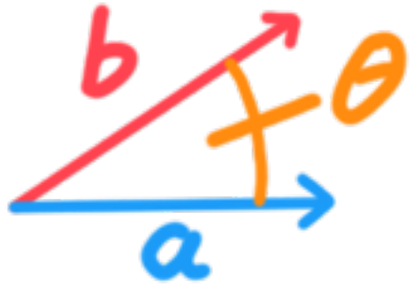
Inverse Kinematik — Jacobian

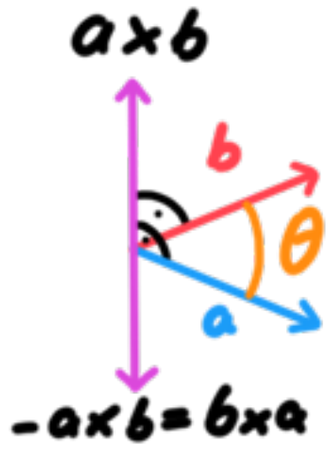


https://www.youtube.com/watch?v=9kTd6MwRoCI

Cyclic Coordinate Descent (CCD)

- Der Positionsfehler des Endeffektors wird schrittweise pro Gelenk reduziert
- Mit dem Skalarprodukt erhält man den Korrekturwinkel für das Gelenk
- Mit dem Kreuzprodukt erhält man die Richtung der Korrektur



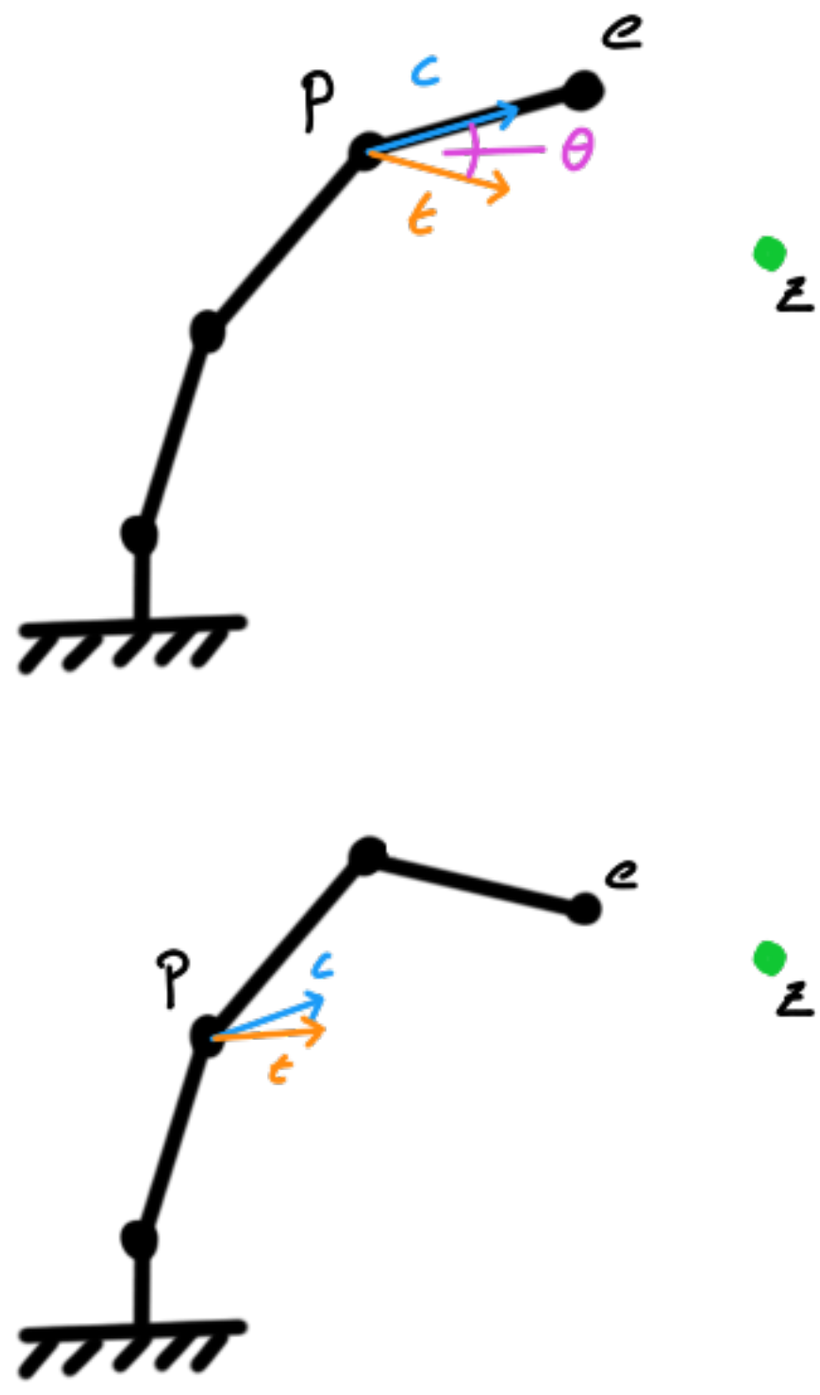


$$a \cdot b = |a||b| \cos\theta$$

$$a \times b = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

Cyclic Coordinate Descent (CCD) — Algorithmus (2D)

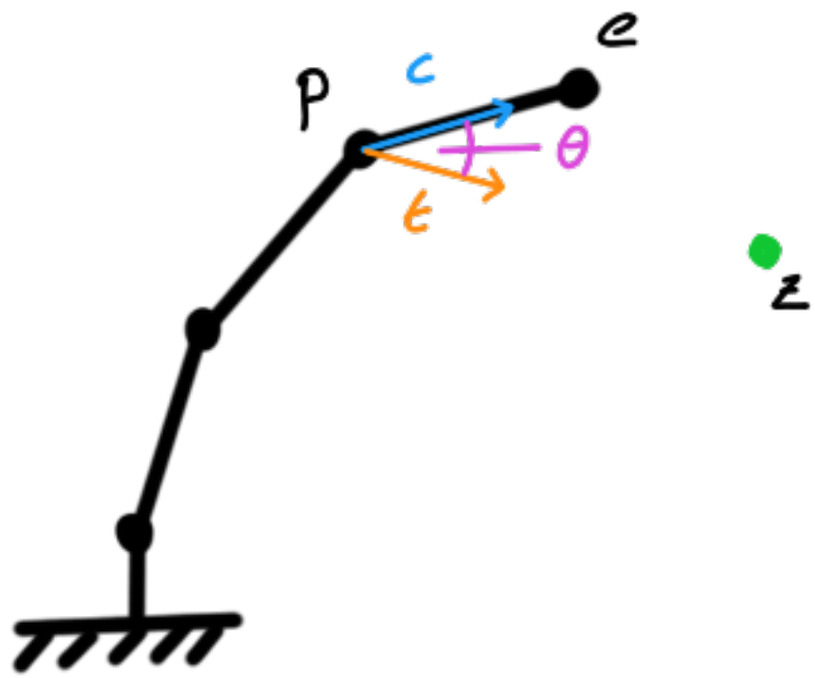
z – Zielposition
e – aktuelle Endeffektorposition
repeat
  for all Gelenke, angefangen beim Endeffektor **do**
    p – aktuelle Position des Gelenks
    e – aktuelle Endeffektorposition
    $c = e - p$
    $t = z - p$
    Normalisiere c und t
    $\cos\theta = c \cdot t$
    if $\cos\theta < 1.0$ **then**
      $k = t \times c$
      if $k.z > 0$ **then**
        $c.orientierung.z = c.orientierung.z - \theta$
      else
        $c.orientierung.z = c.orientierung.z + \theta$
      end if
    end if
  end for
until $|z - e| < \varepsilon$



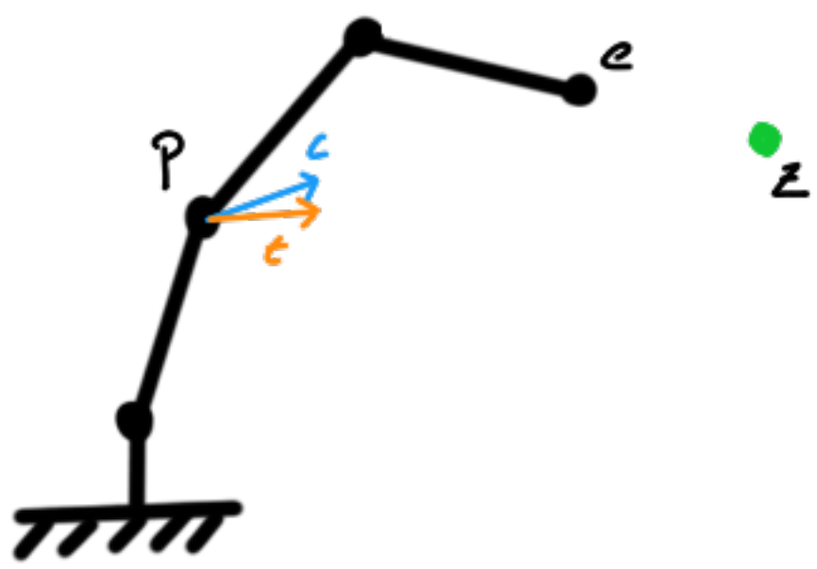
Inverse Kinematik — Cyclic Coordinate Descent

Inverse Kinematik CCD

```
function InverseKinematicsCCD(goal, q)
    base = [0;0;0;1]
    δ = 0.01

    # Endeffektor
    e = A1(q[1]) * A2(q[2]) * A3(q[3]) * base

    # Drittes Segment
    p = A1(q[1]) * A2(q[2]) * base
    c = e[1:2] - p[1:2]
    t = goal - p[1:2]
    c = normalize([c;0])
    t = normalize([t;0])
    cθ = dot(c,t)
    if cθ < 1.0 # 1.0 -> θ = 0
        k = cross(t,c)
        θ = minimum([δ, acos(cθ)])
        if k[3] > 0
            q[3] = q[3] - θ
        else
            q[3] = q[3] + δ
        end
    end
```

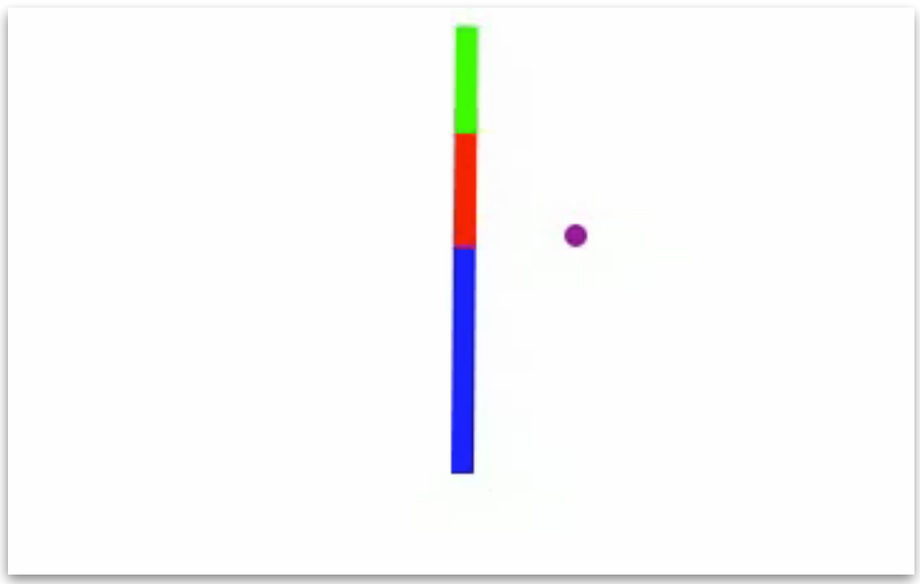


```
    # Zweites Segment
    p = A1(q[1]) * base
    c = e[1:2] - p[1:2]
    t = goal - p[1:2]
    c = normalize([c;0])
    t = normalize([t;0])
    cθ = dot(c,t)
    if cθ < 1.0 # 1.0 -> θ = 0
        k = cross(t,c)
        θ = minimum([δ, acos(cθ)])
        if k[3] > 0
            q[2] = q[2] - θ
        else
            q[2] = q[2] + δ
        end
    end

    # Erstes Segment
    p = base
    c = e[1:2] - p[1:2]
    t = goal - p[1:2]
    c = normalize([c;0])
    t = normalize([t;0])
    cθ = dot(c,t)
    if cθ < 1.0 # 1.0 -> θ = 0
        k = cross(t,c)
        θ = minimum([δ, acos(cθ)])
        if k[3] > 0
            q[1] = q[1] - θ
        else
            q[1] = q[1] + δ
        end
    end
    q
end
```

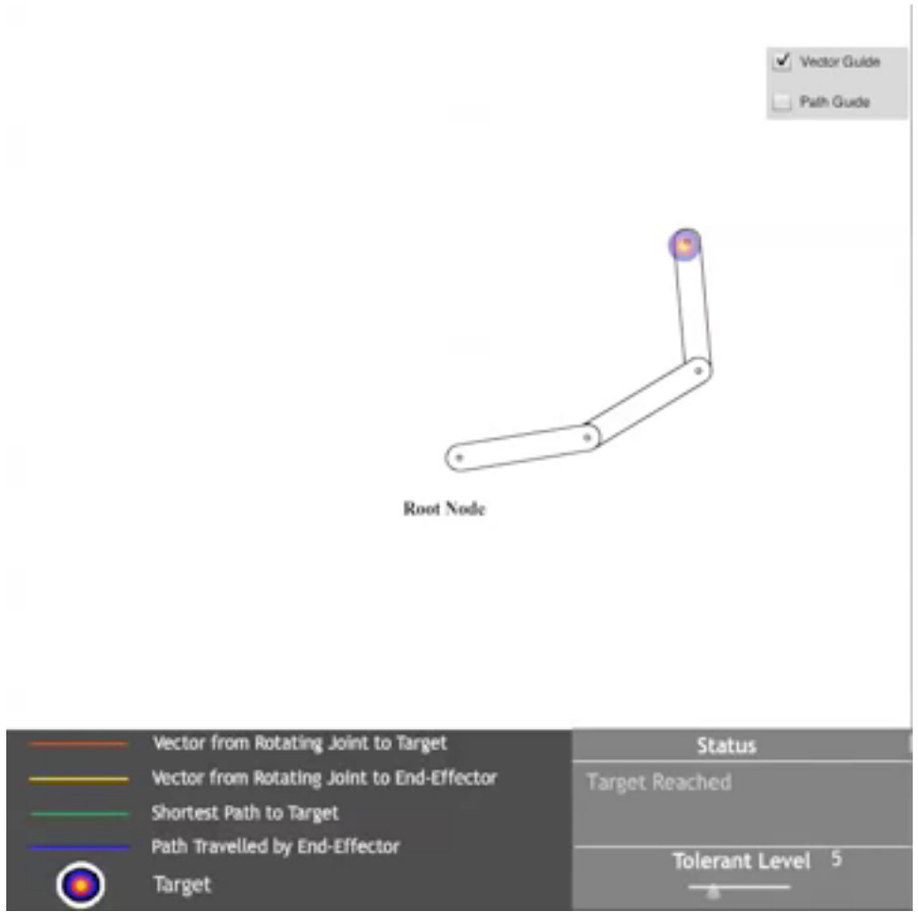


Inverse Kinematik — Cyclic Coordinate Descent

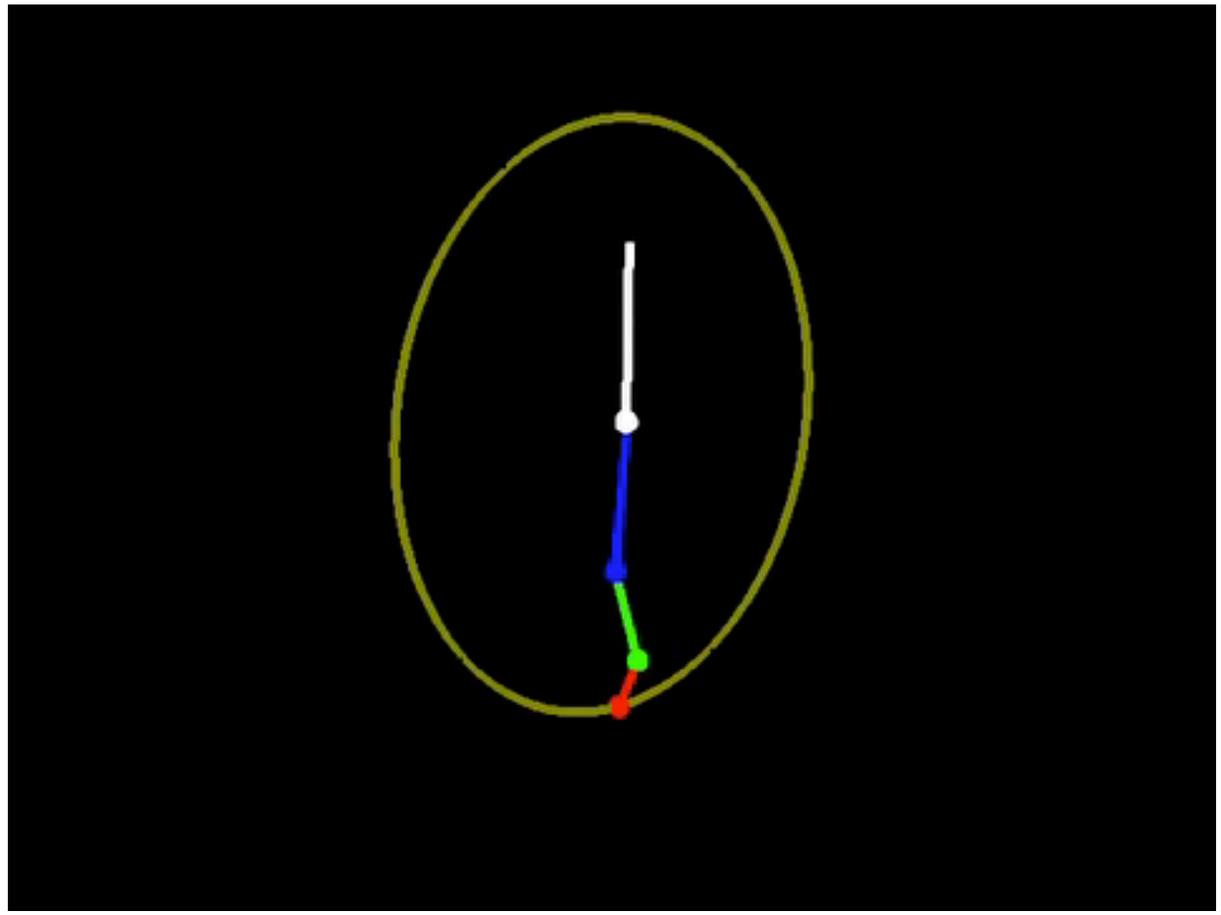
```
q    = [π/2 0.01 0.01]
goal = [1.0; 0.5]
ϵ    = 0.1
j    = 0

ioff()

for i = 1:200
    j = j + 1
    q = InverseKinematicsCCD(goal, q)
    fig = PlotRoboter(q,goal)
    savefig("ikccd/robot_$(j-1).png", bbox_inches="tight")
    close(fig)
end
```



Cyclic Coordinate Descent (CCD)



https://www.youtube.com/watch?v=MvuO9ZHGr6k



https://www.youtube.com/watch?v=tLtKafskg4Q