



## 07. Kinematika, inverz kinematika, Szimulált robotkar programozása csukló-, és munkatérben



### Warning

**ZH2** (Roslaunch, ROS paraméter szerver. Kinematika, inverz kinematika.) és a  
**Kötelező program bemutatás december 6.**

Ismétlés

## 3D transzformációk

•



**Pozíció:** 3 elemű offset vektor

• **Orientáció:** 3 x 3 rotációs matrix

- további orientáció reprezentációk: Euler-szögek, RPY, angle axis, quaternion

• **Helyzet** (pose):  $4 \times 4$  transzformációs mátrix

• **Koordináta rendszer** (frame): null pont, 3 tengely, 3 bázis vektor, jobbkéz-szabály

• **Homogén transzformációk:** rotáció és transláció együtt

- pl.  $\mathbf{R}$  rotáció és  $\mathbf{v}$  transláció esetén:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & v_x \\ r_{2,1} & r_{2,2} & r_{2,3} & v_y \\ r_{3,1} & r_{3,2} & r_{3,3} & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

• **Homogén koordináták:**

- **Vektor:** 0-val egészítjük ki,  $\mathbf{a}_H = \begin{bmatrix} \mathbf{a} \\ 0 \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \\ 0 \end{bmatrix}$
- **Pont:** 1-gyel egészítjük ki,  $\mathbf{p}_H = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$
- Transzformációk alkalmazása egyszerűbb:

$$\mathbf{q} = \mathbf{R}\mathbf{p} + \mathbf{v} \rightarrow \begin{bmatrix} \mathbf{q} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$$

• **Szabadsági fok** (DoF): egymástól független mennyiségek száma.

## Robotikai alapok



- Robotok felépítése: **szegmensek** (segment, link) és **csuklók** (joints)
- **Munkatér** (task space, cartesian space):
  - Háromdimenziós tér, ahol a feladat, trajektóriák, akadályok, stb. definiálásra kerülnek.
  - **TCP** (Tool Center Point): az end effektorhoz rögzített koordináta rendszer (frame)
  - **Base/world frame**
- **Csuklótér** (joint space):
  - A robot csuklóihoz rendelt mennyiségek, melyeket a robot alacsony szintű irányító rendszere értelmezni képes.
  - csukló koordináták, sebességek, gyorsulások, nyomatékok...

## Elmélet

### Kinematika, inverz kinematika

### Def. Kinematika

A TCP (vagy bármilyen más) helyzetének kiszámítása a csukló koordinátákból.

- Kinematikai modell
  - Denavit-Hartenberg (DH) konvenció
  - URDF (Unified Robotics Description Format, XML-alapú)

Ha a segmensekhez rendelt koordináta rendszerek rendre  $(base, 1, 2, 3, \dots, TCP)$ , a szomszédos  $(i)$  and  $(i+1)$  szegmensek közötti transzfomációk  $(T_{i+1,i}(q_{i+1}))$  (mely a közbezárt csukló szögének függvénye), a transzfomáció a base frame és a TCP között felírható  $(n)$  csuklós robotra):

$$T_{TCP,base}(q_1, \dots, q_n) = T_{TCP,n-1}(q_n) \cdot T_{n-1,n-2}(q_{n-1}) \cdot \dots \cdot T_{2,1}(q_2) \cdot T_{1,base}(q_1)$$

### Def. Inverz kinematika

Csukló koordináták kiszámítása a (kívánt) TCP (vagy bármilyen más) pose eléréséhez.

## Differenciális inverz kinematika

### Def. Differenciális inverz kinematika

A csukló koordináták mely változtatása éri el a kívánt, **kis mértékű változást** a TCP helyzetében (rotáció és transláció).

- **Jacobi-mátrix** (Jacobian): egy vektorértékű függvény elsőrendű parciális deriváltjait tartalmazó mátrix.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x_1}{\partial q_1} & \frac{\partial x_1}{\partial q_2} & \frac{\partial x_1}{\partial q_3} & \dots & \frac{\partial x_1}{\partial q_n} \\ \frac{\partial x_2}{\partial q_1} & \frac{\partial x_2}{\partial q_2} & \frac{\partial x_2}{\partial q_3} & \dots & \frac{\partial x_2}{\partial q_n} \\ \frac{\partial x_3}{\partial q_1} & \frac{\partial x_3}{\partial q_2} & \frac{\partial x_3}{\partial q_3} & \dots & \frac{\partial x_3}{\partial q_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_m}{\partial q_1} & \frac{\partial x_m}{\partial q_2} & \frac{\partial x_m}{\partial q_3} & \dots & \frac{\partial x_m}{\partial q_n} \end{bmatrix}$$

- **Jacobi-mátrix jelentősége robotikában:** megadja az összefüggést a csuklósebességek és a TCP sebessége között.

$$\left[ \begin{matrix} \mathbf{v} \\ \mathbf{\omega} \end{matrix} \right] = \mathbf{J}(\mathbf{q}) \dot{\mathbf{q}}$$

### Inverz kinematika Jacobi inverz felhasználásával

1. Számítsuk ki a kívánt és az aktuális pozíció különbségét:  $\Delta \mathbf{r} = \mathbf{r}_{\text{desired}} - \mathbf{r}_0$
2. Számítsuk ki a rotációk különbségét:  $\Delta \mathbf{R} = \mathbf{R}_{\text{desired}} \mathbf{R}_0^T$ , majd konvertáljuk át axis angle reprezentációba  $(\mathbf{t}, \phi)$
3. Számítsuk ki  $\Delta \mathbf{q} = \mathbf{J}^{-1}(\mathbf{q}_0) \cdot \left[ \begin{matrix} \mathbf{k}_1 \cdot \Delta \mathbf{r} \\ \mathbf{k}_2 \cdot \phi \cdot \mathbf{t} \end{matrix} \right]$ , ahol az inverz lehet pseudo-inverz, vagy transzponált
4.  $\mathbf{q}_{\text{better}} = \mathbf{q}_0 + \Delta \mathbf{q}$

## Gyakorlat

### 1: Install rrr-arm

1. Telepítsük a dependency-ket.

```
sudo apt update
sudo apt-get install libpoco-dev
sudo apt-get install ros-foxy-control-msgs ros-foxy-realtime-tools ros-foxy-xacro ros-foxy-joint-state-publisher-gui
pip3 install kinpy
```

#### Tip

A `kinpy` csomag forrását is töltsük le, hasznos lehet az API megértése szempontjából: <https://pypi.org/project/kinpy/>

2. Clone-ozzuk és build-eljük a repo-t.

```

mkdir -p ~/doosan2_ws/src
cd ~/doosan2_ws/src
git clone https://github.com/TamasDNagy/doosan-robot2.git
git clone https://github.com/ros-controls/ros2_control.git
git clone https://github.com/ros-controls/ros2_controllers.git
git clone https://github.com/ros-simulation/gazebo_ros2_control.git
cd ros2_control && git reset --hard 3dc62e28e3bc8cf636275825526c11d13b554bb6
&& cd ..
cd ros2_controllers && git reset --hard 83c494f460f1c8675f4fdd6fb8707b87e81cb197
&& cd ..
cd gazebo_ros2_control && git reset --hard
3dfe04d412d5be4540752e9c1165ccf25d7c51fb && cd ..
git clone -b ros2 --single-branch https://github.com/ros-planning/moveit_msgs
cd ~/doosan2_ws
rosdep update
rosdep install --from-paths src --ignore-src --rosdistro foxy -r -y
colcon build --cmake-args -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
. install/setup.bash
rosdep update

```

Adjuk hozzá az alábbi sort a `~/doosan2_ws/.bashrc` fájlhoz:

```
source ~/doosan2_ws/install/setup.bash
```

3. Teszteljük a szimulátort, új terminál ablakokban:

```

#ros2 launch dsr_launcher2 single_robot_rviz.launch.py model:=a0912 color:=blue
ros2 launch dsr_launcher2 single_robot_rviz_topic.launch.py model:=a0912 color:=blue

```

4. Állítsuk elő a robotot leíró urdf fájlt: TODO

```

cd ~/catkin_ws/src/rrr-arm/urdf
roslaunch xacro xacro rrr_arm.xacro > rrr_arm.xacro.urdf

```

## 2: Robot mozgatása csuklótérben

1. Iratkozzunk fel a robot csuklósögeit (konfigurációját) publikáló topicra.  
Hozzunk létre publisher-eket a csuklók szögeinek beállítására használható topic-okhoz.

### Warning

A Kinpy és a ROS nem mindig azonos sorrendben kezeli a csuklószokeket. Az alábbi két sorrend fordul elő: **1. [gripper\_joint\_1, gripper\_joint\_2, joint\_1, joint\_2, joint\_3, joint\_4]** - /rrr\_arm/joint\_states topic - `kp.jacobian.calc_jacobian(...)` függvény

**2. [joint\_1, joint\_2, joint\_3, joint\_4, gripper\_joint\_1, gripper\_joint\_2]** - `chain.forward_kinematics(...)` függvény - `chain.inverse_kinematics(...)` függvény

2. Mozgassuk a robotot [1.0, 1.0, 1.5, 1.5] konfigurációba.

## 3. Kinematika

1. Importáljuk a `kinpy` csomagot és olvassuk be a robotot leíró urdf fájlt:

```
import kinpy as kp

chain = kp.build_serial_chain_from_urdf(open("/home/<USERNAME>/catkin_ws/src/rrr-arm/urdf/rrr_arm.xacro.urdf").read(), "gripper_frame_cp")
print(chain)
print(chain.get_joint_parameter_names())
```

2. Számítsuk ki, majd irassuk ki a TCP pozícióját az adott konfigurációban a `kinpy` csomag segítségével. A <https://pypi.org/project/kinpy/> oldalon lévő példa hibás, érdemes az alábbi példa kódból kiindulni:

```
th1 = np.random.rand(2)
tg = chain.forward_kinematics(th1)
th2 = chain.inverse_kinematics(tg)
self.assertTrue(np.allclose(th1, th2, atol=1.0e-6))
```

## 4: Inverz kinematika Jacobi inverz módszerrel

Írjunk metódust, amely az előadásban bemutatott Jakobi inverz módszerrel valósítja meg az inverz kinematikai feladatot a roboton. Az orientációt hagyjuk



figyelman kívül. Mozcassuk a TCP-t a `(0.59840159, -0.21191189, 0.42244937)` pozícióba.

1. Írjunk egy ciklust, melynek megállási feltétele a `delta_r` megfelelő nagysága, vagy `rospy.is_shutdown()`.
2. Számítsuk ki a kívánt és a pillanatnyi TCP pozíciók különbségét (`delta_r`). Skálázzuk `k_1` konstanssal.
3. `phi_dot_t` legyen `[0.0, 0.0, 0.0]` (ignoráljuk az orientációt).
4. Konkaténáljuk `delta_r` és `phi_dot_t`-t.
5. Számítsuk ki a Jacobi mátrixot az adott konfigurációban a `kp.jacobian.calc_jacobian(...)` függvény segítségével.
6. Számítsuk ki Jacobi mátrix pszeudo-inverzét `np.linalg.pinv(...)`.
7. A fenti képlet segítségével számítsuk ki `delta_q`-t.
8. Növeljük a csuklózögeket a kapott értékekkel.

### *Bónusz:* Inverz kinematika orientációval

Egészítsük ki az előző feladat megoldását úgy, hogy az orientációt is figyelembe vesszük az inverz kinematikai számítás során.

## Hasznos linkek

- [doosan-robot2 github](#)
- <https://pypi.org/project/kinpy/>

- [https://en.wikipedia.org/wiki/Axis%E2%80%93angle\\_representation](https://en.wikipedia.org/wiki/Axis%E2%80%93angle_representation)
- <https://www.rosroboticslearning.com/jacobian>