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



Ismétlés

•



Pozíció: 3 elemű offszet vektor

- Orientáció: 3 x 3 rotációs matrix
  - további orientáció reprezentációk: Euler-szögek, RPY, angle axis, quaternion
- **Helyzet** (pose): 4 × 4 transzformációs mártrix
- **Koordináta rendszer** (frame): null pont, 3 tengely, 3 bázis vektor, jobbkézszabály
- Homogén transzformációk: rotáció és transzláció együtt
  - pl. \(\mathbf{R}\) rotáció és \(\mathbf{v}\) transzláció esetén:

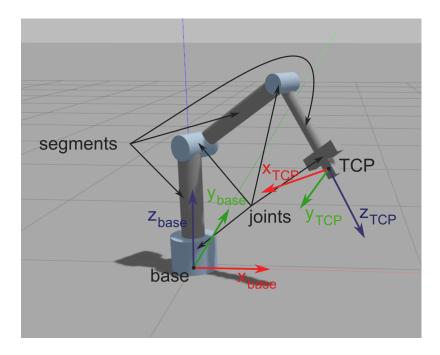
 $$$ \mathbf{T} = \left[ \mathbf{R} & \mathbf{0} & 1 \right] = \left[ \mathbf{T}_{1,1} & r_{1,2} & r_{1,3} & v_x \right] = \left[ \mathbf{T}_{3,1} & r_{3,2} & r_{3,3} & v_x \right] \\ v_y \left[ 3,1 \right] & r_{3,2} & r_{3,3} & v_x \right] \\$ 

- · Homogén koordináták:
  - Vektor: 0-val egészítjük ki, \(\mathbf{a\_H}=\left[\matrix{\mathbf{a} \\ 0}\right]=\left[\matrix{a x \\ a y \\ a z \\ 0}\right]\)
  - Pont: 1-gyel egészítjük ki, \(\mathbf{p\_H}=\left[\matrix{\mathbf{p} \\ 1}\right]=\left[\matrix{p\_x \\ p\_y \\ p\_z \\ 1}\right]\)
  - Transzformációk alkalmazása egyszerűbb:

 $$$ \left( \mathbf{q} = \mathbf{R}\right) + \mathbf{v} \to \left[\mathbf{q} \right] + \mathbf{q} \\ \left( \mathbf{q} \right) = \left[\mathbf{q} \right] + \mathbf{q} \\ \left( \mathbf{q} \right) + \mathbf{q} \\ \left( \mathbf{q}$ 

• 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

#### Kinematika



### Def. Kinematika

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

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

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

#### Inverz kinematika



### Def. Inverz kinematika

Csukló koordináták kiszámítása a (kívánt) TCP (vagy bármi 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 transzláció).

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

 $$$ \prod_{1} \left[ \frac{x_1}{\left(x_1} \right] & \frac{x_1}{\left(x_1\right)} & \frac{x_2}{\left(x_1\right)} & \frac{x_2}{\left(x_1\right)} & \frac{x_2}{\left(x_2\right)} & \frac{x_2}{\left(x$ 

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

```
 $$ \left[ \left[ \operatorname{\mathcal D} \right] \right] = \mathbb{J} (\mathbb{q}) \cdot \mathbb{q} \]
```

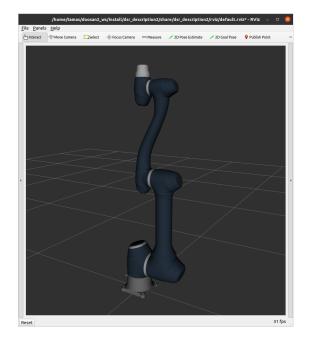
## Inverz kinematika Jacobi inverz felhasználásával

- 1. Számítsuk ki a kívánt és az aktuális pozíció különbségét:  $\label{eq:r} = \mathbf{r}_{\text{o}} \mathbf{r}_0$
- 2. Számítsuk ki a rotációk különbségét:  $\(\Delta R) = \mathbb{R}$  \_{desired}\mathbf{R}\_{0}^{T}\), majd konvertáljuk át axis angle reprezentációba  $\((\Delta R))$ \)
- 4.  $\mbox{\mbox{\mbox{$\langle$}} q} {\mbox{\mbox{$\langle$}} q} = \mbox{\mbox{$\langle$}} q} {\mbox{\mbox{$\langle$}} q} + \mbox{\mbox{$\langle$}} q} {\mbox{\mbox{$\langle$}}}$

# Gyakorlat

### 1: Doosan2 install

- 1. Állítsuk vissza a ~/.bashrc fájlt ROS2 alapértelmezettre.
- 2. 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-foxyjoint-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/

## 3. 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 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



## Warning

A VM-eken már telepítve van, de itt is frissítsük a repo-t:

```
cd ~/doosan2_ws/src/doosan-robot2
git pull
cd ~/doosan2_ws
colcon build --cmake-args -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
```

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

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

4. Teszteljük a szimulátort, új teminál ablakban:

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

## 2: Robot mozgatása csuklótérben

1. Hozzunk létre új python forrásfájlt doosan2\_controller.py névvel a ~/ros2\_ws/src/ros2\_course/ros2\_course mappában. Adjuk meg az új entry point-ot a setup.py-ban a megszokott módon. Iratkozzunk fel a robot csuklószögeit (konfigurációját) publikáló topicra. Hozzunk létre publisher-t a csuklók szögeinek beállítására használható topic-hoz.

```
/joint_states
/joint_cmd
```

2. Mozgassuk a robotot q = [0.24, -0.3, 1.55, 0.03, 1.8, 0.5] konfigurációba.

## 3. Kinematika

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

```
import kinpy as kp

self.chain = kp.build_serial_chain_from_urdf(open(
    "/home/<USERNAME>/doosan2_ws/src/doosan-robot2/dsr_description2/urdf/
a0912.blue.urdf").read(),
    "link6")
print(self.chain.get_joint_parameter_names())
print(self.chain)
```

2. Számítsuk ki, majd irassuk ki a TCP pozícióját az adott konfigurációban a kinpy csomag segítségével.

```
tg = chain.forward\_kinematics(th1)
```

## 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 figyelmen kívül. Mozgassuk a TCP-t a (0.55, 0.05, 0.45) pozícióba. Ábrázoljuk a TCP trajektóriáját Matplotlib segítségével.

- 1. Írjunk egy ciklust, melynek megállási feltétele a delta\_r megfelelő nagysága és rclpy.ok() .
- 2. Számítsuk ki a kívánt és a pillanatnyi TCP pozíciók különbségét (delta\_r). Skálázzuk k\_1 konstanssal.
- 3. omega legyen [0.0, 0.0, 0.0] (ignoráljuk az orientációt).
- 4. Konkatenáljuk delta\_r és omega-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 számítsük ki delta\_q-t.
- 8. Növeljük a csuklószögeket 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/
- $\bullet\ https://en.wikipedia.org/wiki/Axis\%E2\%80\%93 angle\_representation$
- https://www.rosroboticslearning.com/jacobian