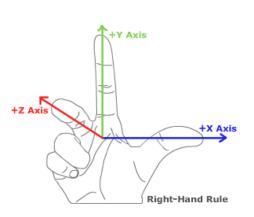
07. Kinematika, inverz kienamtika, Szimulált robotkar programozása csukló-, és munkatérben

Ismétlés

3D transzformációk

•



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. $\mbox{\mbox{$\backslash$}}\$ rotáció és $\mbox{\mbox{\mbox{$\backslash$}}}\$ 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] \\$

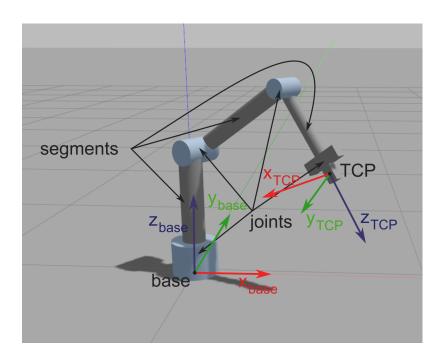
• Homogén koordináták:

- 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, $\ (\mathbf{p_H} = \left[\mathbf{p_H} \right] \ 1 \right] = \left[\mathbf{p_x \ p_y \ p_z \ 1 \right]$
- Transzformációk alkalmazása egyszerűbb:

 $$$ \left\{ q = \mathbb{R}\mathbb{q} + \mathbb{q} \right] $$ \left(mathbf\{q\} + \mathbb{q} \right) = \left[\mathbb{R} & \mathbb{q} \right] $$ \left(mathbf\{q\} \ \ \right] \right] $$ \left[\mathbb{q} \ \ \right] $$ \left$

• 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 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):

$$\begin{split} & \text{$T_{TCP,base}(q_1, \cdot q_n) = T_{TCP,n-1}(q_{n}) \cdot T_{n-1,n-2} \\ & (q_{n-1}) \cdot T_{2,1}(q_2) \cdot T_{1,base}(q_1) \cdot S_{n-1,n-2} \\ & (q_{n-1}) \cdot T_{2,1}(q_2) \cdot T_{1,base}(q_1) \cdot S_{n-1,n-2} \\ & (q_{n-1}) \cdot T_{n-1,n-2} \\ & (q$$



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.

```
 $$ \left[ \mathbf{J} = \left[ \mathbf x_1 \right] \right] & \frac{partial x_1}{\operatorname{q_3} & \operatorname{partial x_1}_{\operatorname{q_3} & \operatorname{partial x_1}_{\operatorname{q_3} & \operatorname{partial x_1}_{\operatorname{q_3} & \operatorname{partial x_2}_{\operatorname{q_3} & \operatorname{partial x_3}_{\operatorname{q_3} & \operatorname{partial x_3}_{\operatorname{q_3} & \operatorname{partial x_3}_{\operatorname{q_3} & \operatorname{partial x_3}_{\operatorname{q_3} & \operatorname{q_3} & \operatorname{partial x_3}_{\operatorname{q_3} & \operatorname{partial x_3}_{\operatorname{q_3} & \operatorname{partial x_3}_{\operatorname{q_1} & \operatorname{q_3} & \operatorname{partial x_3}_{\operatorname{q_1} & \operatorname{q_3} & \operatorname{q_1}_{\operatorname{q_1} & \operatorname{q_3} & \operatorname{q_1}_{\operatorname{q_1} & \operatorname{q_1}_{\operatorname{q_2} & \operatorname{q_1}_{\operatorname{q_1} & \operatorname{q_2}_{\operatorname{q_1} & \operatorname{q_1}_{\operatorname{q_2} & \operatorname{q_1}_{\operatorname{q_1} & \operatorname{q_2}_{\operatorname{q_1} & \operatorname{q_1}_{\operatorname{q_2} & \operatorname{q_1}_{\operatorname{q_1} & \operatorname{q_2}_{\operatorname{q_1} & \operatorname{q_1}_{\operatorname{q_2} & \operatorname{q_1}_{\operatorname{q_1} & \operatorname{q_1}_{\operatorname{q_1}} & \operatorname{q_1}_{\operatorname{q_1}_{\operatorname{q_1} & \operatorname{q_1}_{\operatorname{q_1}} & \operatorname{q_1}_{\operatorname{q_1}}}}}}}}}}}}}}}}}
```

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

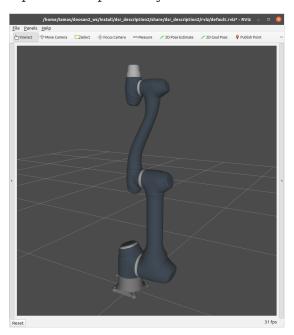
Inverz kinematika Jacobi inverz felhasználásával

- 1. Számítsuk ki a kívánt és az aktuális pozíció különbségét: $\ \$ = $\$ \mathbf{r} {desired} \mathbf{r} 0\)
- 2. Számítsuk ki a rotációk különbségét: $\(\Delta R) = \mathbb{R}$ _{desired}\mathbf{R}_{0}^{T}\), majd konvertáljuk át axis angle reprezentációba \((\mathbf{t},\phi)\)
- 3. Számítsuk ki \(\Delta\mathbf{ q}=\mathbb{J}^{-1}(\mathbb{q_0})\cdot \left\{ \frac{k_1 \cdot k_2 \cdot k_2 \cdot k_1 \cdot k_1 \cdot k_2 \cdot k_1 \cdot k_1 \cdot k_2 \cdot k_1 \cdot k_2 \cdot k_1 \cdot k_2 \cdot k_1 \cdot k_1 \cdot k_2 \cdot k_1 \cdot k_1 \cdot k_1 \cdot k_2 \cdot k_1 \cdot k_1 \cdot k_1 \cdot k_2 \cdot k_1 \cdot
- 4. $\mbox{\mbox{$\langle q \rangle {better} = \mathbb{q} {0} + \mathbb{q} {q} }}$

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-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
```

Warning

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

```
cd \sim/doosan2_ws/src/doosan-robot2 git pull cd \sim/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, 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. Konkatenáljuk 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 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/
- https://en.wikipedia.org/wiki/Axis%E2%80%93angle_representation
- https://www.rosroboticslearning.com/jacobian