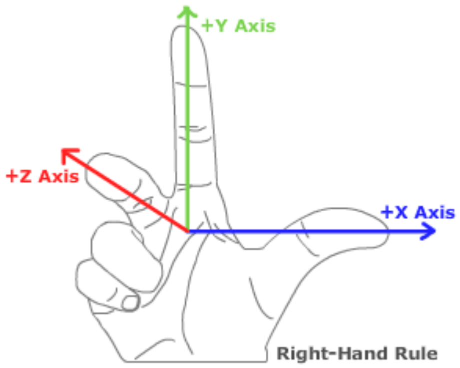
07. Kinematika, inverz kienamtika, Szimulált robotkar programozása csukló-, és munkatérben

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, jobbkéz-szabály
- **Homogén transzformációk:** rotáció és transzláció együtt
  - pl. $\mathbf{R}$ rotáció és $\mathbf{v}$ transzláció esetén:

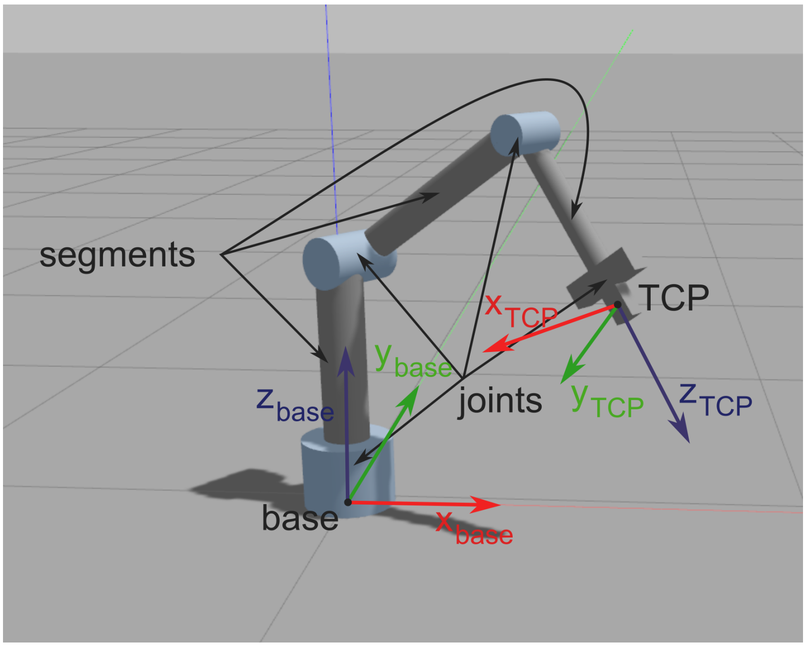
$$\mathbf{T} = \left[\begin{matrix}\mathbf{R} & \mathbf{v}\\\mathbf{0} & 1 \end{matrix}\right] = \left[\begin{matrix}r_{1,1} & r_{1,2} & r_{1,3} & v_x\\r_{2,1} & r_{2,2} & r_{2,3} & v_y\\r_{3,1} & r_{3,2} & r_{3,3} & v_z\\ 0 & 0 & 0 & 1 \end{matrix}\right]$$

- **Homogén koordináták:**
  - **Vektor:** 0-val egészítjük ki, $\mathbf{a_H}=\left[\begin{matrix}\mathbf{a} \\ 0\end{matrix}\right]=\left[\begin{matrix}a_x \\ a_y \\ a_z \\ 0\end{matrix}\right]$
  - **Pont:** 1-gyel egészítjük ki, $\mathbf{p_H}=\left[\begin{matrix}\mathbf{p} \\ 1\end{matrix}\right]=\left[\begin{matrix}p_x \\ p_y \\ p_z \\ 1\end{matrix}\right]$
  - Transzformációk alkalmazása egyszerűbb:

$$\mathbf{q} = \mathbf{R}\mathbf{p} + \mathbf{v} \to \left[\begin{matrix}\mathbf{q} \\ 1\end{matrix}\right] = \left[\begin{matrix}\mathbf{R} & \mathbf{v}\\\mathbf{0} & 1 \end{matrix}\right] \left[\begin{matrix}\mathbf{p} \\ 1\end{matrix}\right]$$

- **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 (DH) 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):

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

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.

  \[\mathbf{J} = \left[\matrix{\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} \\}\right] \]

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

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

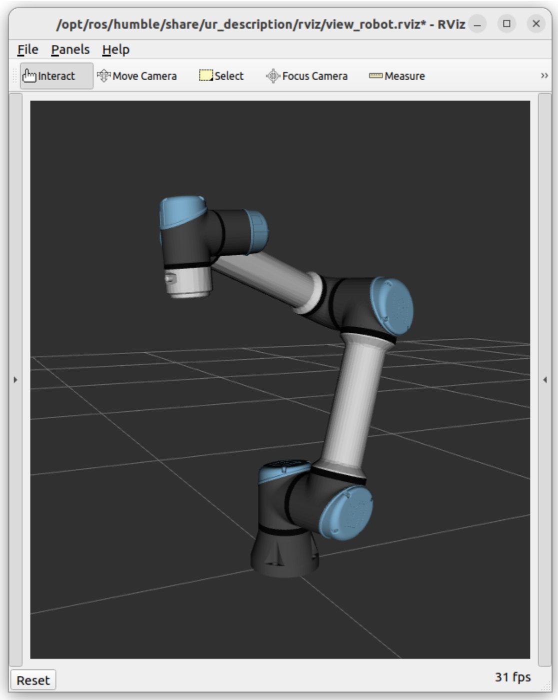
Inverz kinematika Jacobi inverz felhasználásával

1. Számítsuk ki a kívánt és az aktuális pozíció különbségét: $\Delta\mathbf{r} = \mathbf{r}_{desired} - \mathbf{r}_0$
2. Számítsuk ki a rotációk különbségét: $\Delta\mathbf{R} = \mathbf{R}_{desired}\mathbf{R}_{0}^{T}$, majd konvertáljuk át axis angle reprezentációba $(\mathbf{t},\phi)$
3. Számítsuk ki $\Delta\mathbf{ q}=\mathbf{J}^{-1}(\mathbf{q_0})\cdot \left[\matrix{k_1 \cdot \Delta\mathbf{r} \\ k_2 \cdot \mathbf{\omega}}\right]$, ahol az inverz lehet pszeudo-inverz, vagy transzponált
4. $\mathbf{q}_{better} = \mathbf{q}_{0} + \Delta\mathbf{q}$

Gyakorlat

1: UR install

1. Telepítsük a dependency-ket és a UR driver-t.




```
sudo apt update
sudo apt upgrade
sudo apt-get install ros-humble-ur python3-pip
pip3 install kinpy
```

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

2. Moodle-ről töltsük le a forrásfájlokatokat tartalmazó zip-et (`ur_ros2_course.zip`). A `view_ur.launch.py` fájlt másoljuk a `ros2_course/launch` mappába, a `topic_latcher.py` fájlt pedig a `ros2_course/ros2_course` mappába. Adjuk hozzá az alábbi sorokat a `setup.py` fájlhoz (launch és entry point):

```
import os
from glob import glob

# ...

data_files=[
    ('share/ament_index/resource_index/packages',
        ['resource/' + package_name]),
    ('share/' + package_name, ['package.xml']),
    # Include all launch files.
    (os.path.join('share', package_name),
        glob('launch/*launch.[pxy][yma]*'))
```

```
],

# ...

entry_points={
'console_scripts': [
    # ...
    'topic_latcher = ros2_course.topic_latcher:main',
],
```

3. Indítsuk el a szimulátort, mozgassuk a csuklókat a Joint State Publisher GUI segítségével.

```
ros2 launch ros2_course view_ur.launch.py ur_type:=ur5e
```

> **Tip**
>
> Próbáljunk ki más robotokat is a `ur_type` argumentum beállításával (ur3, ur3e, ur5, ur5e, ur10, ur10e, ur16e, ur20)

2: Robot mozgatása csuklótérben

1. Hozzunk létre új python forrásfájlt `ur_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
/set_joint_states
```

2. Mozgassuk a robotot `q = [-1.28, 4.41, 1.54, -1.16, -1.56, 0.0]` konfigurációba.

3. Kinematika

1. A szimulátor egy topicban publikálja a robotot leíró urdf-t. Iratkozzunk fel erre a topic-ra.

   ```
   /robot_description_latch
   ```

2. Importáljuk a `kinpy` csomagot és hozzuk létre a kinematikai láncot a robotot leíró urdf alapján az előbb implementált callback függvényben:

   ```
   import kinpy as kp

   # ...

   self.chain = kp.build_serial_chain_from_urdf(self.desc, 'tool0')
   print(self.chain.get_joint_parameter_names())
   print(self.chain)
   ```

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

   ```
   p = chain.forward_kinematics(q)
   ```

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 `(0.50, -0.60, 0.20)` pozícióba.

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.

Ábrázoljuk a TCP trajektóriáját Matplotlib segítségével.

````
```python
import matplotlib.pyplot as plt

# ...

# Plot trajectory
ax = plt.figure().add_subplot(projection='3d')
ax.plot(x, y, z, label='TCP trajectory',  ls='-', marker='.')
ax.legend()
ax.set_xlabel('x [m]')
ax.set_ylabel('y [m]')
ax.set_zlabel('z [m]')
plt.show()
```

---
````

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