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

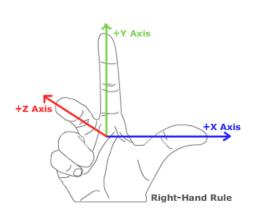
Warning

ZH2 (Roslaunch, ROS paraméter szerver. Kinematika, inverz kinematika.) és a Kötelező program bemutatás május 9.

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. \(\mathbf{R}\) 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] \\$

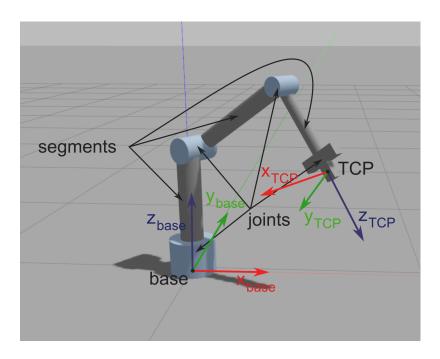
· Homogén koordináták:

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

 $$$ \left(\mathbf{q} = \mathbf{R}\right) + \mathbf{v} \to \left(\mathbf{q} \right) = \left(\mathbf{R}\right) + \mathbf{v} \in \left(\mathbf{q} \right) = \left(\mathbf{R} & \mathbf{v}\right) \in \left(0\} & 1 \right) \left(\mathbf{q} \right) + \left(\mathbf{q} \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



Def. Kinematika

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



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.

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

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

Inverz kinematika Jacobi inverz felhasználásával

- 1. Számítsuk ki a kívánt és az aktuális pozíció különbségét: $\l \$ = $\$ \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_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_2 \cdot k_1 \cdot
- 4. $(\mathbb{q} \{better\} = \mathbb{q} \{0\} + \mathbb{q} \}$

Gyakorlat

1: Install rrr-arm

1. Telepítsük a dependency-ket.

```
sudo apt update
sudo apt-get install ros-noetic-effort-controllers
sudo apt-get install ros-noetic-position-controllers
sudo apt-get install ros-noetic-gazebo-ros-pkgs
sudo apt-get install ros-noetic-gazebo-ros-control
sudo apt-get install ros-noetic-gazebo-ros
pip3 install kinpy
rosdep update
```



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.

```
 \begin{array}{lll} cd & \sim / catkin_ws/src \\ git clone & https://github.com/Robotawi/rrr-arm.git \\ cd & .. \\ catkin & build \\ \end{array}
```

3. Teszteljük a szimulátort, új teminál ablakokban:

roslaunch rrr arm view arm gazebo control empty world.launch

rostopic pub /rrr_arm/joint1_position_controller/command std_msgs/Float64 "data: 1.0" & rostopic pub /rrr_arm/joint2_position_controller/command std_msgs/Float64 "data: 1.0" & rostopic pub /rrr_arm/joint3_position_controller/command std_msgs/Float64 "data: 1.5" & rostopic pub /rrr_arm/joint4_position_controller/command std_msgs/ Float64 "data: 1.5"



Tip

A szimulátor panaszkodni fog, hogy "No p gain specified for pid...", de ez nem okoz gondot a működésében.

4. Állítsuk elő a robotot leíró urdf fájlt:

```
cd ~/catkin_ws/src/rrr-arm/urdf
rosrun xacro xacro rrr arm.xacro > rrr arm.xacro.urdf
```

2: Robot mozgatása csuklótérben

Iratkozzunk fel a robot csuklószögeit (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 csuklószögeket. 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 figyelmen kívül. Mozgassuk a TCP-t a (0.59840159, -0.21191189, 0.42244937) pozícióba.

- 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, majd 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

- rrr-arm model
- https://pypi.org/project/kinpy/
- $\bullet\ https://en.wikipedia.org/wiki/Axis\%E2\%80\%93 angle_representation$
- https://www.rosroboticslearning.com/jacobian