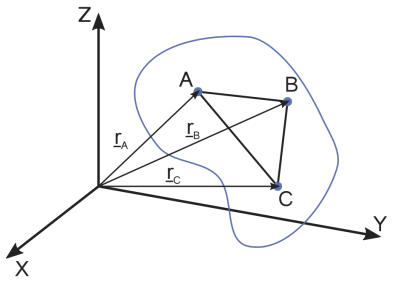
05. Robotikai alapfogalmak, da Vinci sebészrobot programozása szimulált környezetben

Elmélet

Warning

ZH1 (ROS alapok, publisher, subscriber. Python alapok. Robotikai alapfogalmak.) **március 21.**

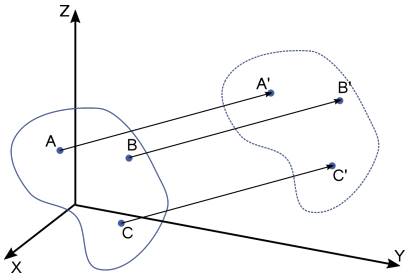
Merev test mozgása



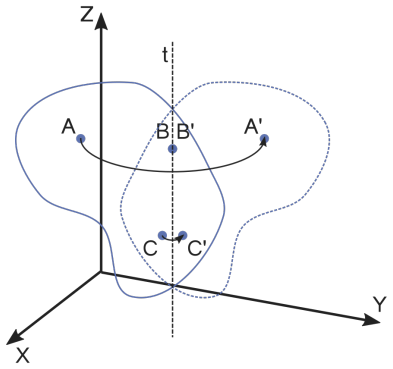
Def. Merev test

Merevnek tekinthető az a test, mely pontjainak távolsága mozgás során nem változik, vagyis bármely két pontjának távolsága időben állandó.

- Merev test alakja, térfogata szintén állandó.
- Merev test térbeli helyzete megadható bármely 3 nem egy egyenesbe eső pontjának helyzetével.

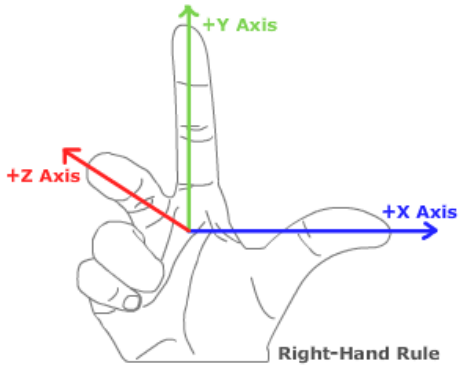


- A test **helyzetét** szemléletesebben megadhatjuk egy tetszőleges pontjának 3 koordinátájával (pozíció) és a test orientációjával.
- Merev testek mozgásai két elemi mozgásfajtából tevődnek össze: **haladó mozgás (transzláció)** és **tengely körüli forgás (rotáció)**
- **Transzlációs mozgás** során a test minden pontja egymással párhuzamos, egybevágó pályát ír le, a test orientációja pedig nem változik.



- **Rotáció** során a forgástengelyen lévő pontok pozíciója nem változik, a test többi pontja pedig a forgástengelyre merőleges síkokban körpályán mozog.
- A **merev test szabad mozgása** is leírható mint egyidejűleg egy bizonyos **tengely körüli forgás és egy haladó mozgás**.

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:

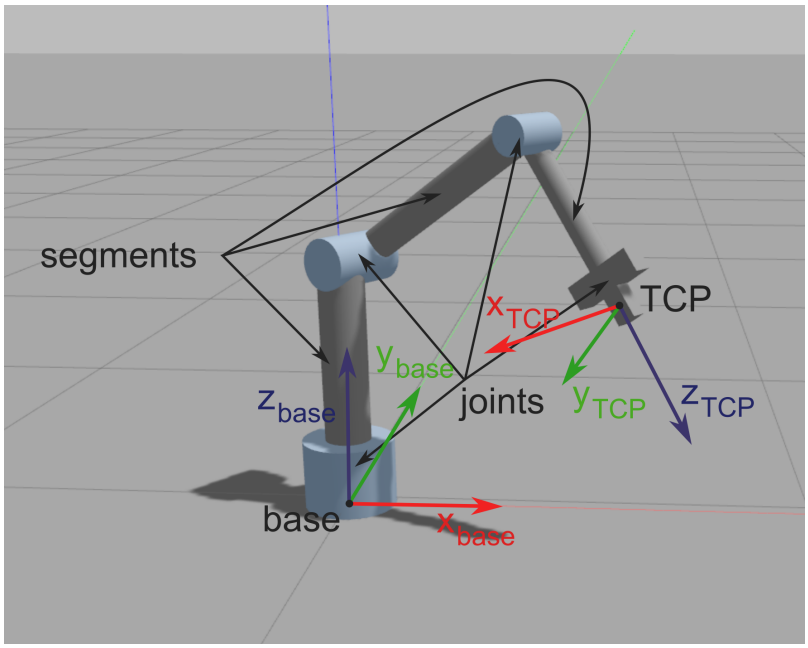
$$ \mathbf{T} = \left[\matrix{\mathbf{R} & \mathbf{v}\\\mathbf{0} & 1 }\right] = \left[\matrix{r_{1,1} & r_{1,2} & r_{1,3} & v_x\\r_{2,1} & r_{2,2} & r_{2,3} & v_y\\r_{3,1} & r_{3,2} & r_{3,3} & v_z\\ 0 & 0 & 0 & 1 }\right] $$

- **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:

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

- **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...

Python libraries

Numpy

- Python library

- High dimension arrays and matrices
- Mathematical functions

```
import numpy as np

# Creating ndarrays
a = np.zeros(3)
a.shape
a.shape=(3,1)
a = np.ones(5)
a = np.empty(10)
l = np.linspace(5, 10, 6)
r = np.array([1,2])    # ndarray from python list
r = np.array([[1,2],[3,4]])
type(r)

# Indexing
l[0]
l[0:2]
l[-1]
r[:,0]

# Operations on ndarrays
r_sin = np.sin(r)
np.max(r)
np.min(r)
np.sum(r)
np.mean(r)
np.std(r)

l < 7
l[l < 7]
np.where(l < 7)

p = np.linspace(1, 5, 6)
q = np.linspace(10, 14, 6)

s = p + q
s = p * q
s = p * 10
s = p + 10
s = p @ q    # dot product
s = r.T
```

If not installed:

```
pip3 install numpy
```

Matplotlib

- Visualization in python
- Syntax similar to Matlab

```
import numpy as np
from matplotlib import pyplot as plt

X = np.linspace(-np.pi, np.pi, 256)
C, S = np.cos(X), np.sin(X)

plt.plot(X, C)
plt.plot(X, S)

plt.show()
```

If not installed:

```
pip3 install matplotlib
```

Gyakorlat

1. dVRK install

1. Ubuntu 20.04-en az alábbi csomagokra lesz sükség:

```
sudo apt install libxml2-dev libraw1394-dev libncurses5-dev qtcreator swig sox espeak
cmake-curses-gui cmake-qt-gui git subversion gfortran libcppunit-dev
libqt5xmlpatterns5-dev python3-wstool python3-catkin-tools python3-osrf-pycommon
ros-noetic-rviz
```

2. Töltsük le és telepítsük a dVRK-t (da Vinci Reserach Kit):

```
cd ~/catkin_ws                    # go in the workspace
wstool init src                   # we're going to use wstool to pull all the code from github
catkin config --cmake-args -DCMAKE_BUILD_TYPE=Release # all code should be
compiled in release mode
cd src                            # go in source directory to pull code
wstool merge https://raw.githubusercontent.com/jhu-dvrk/dvrk-ros/master/
dvrk_ros.rosinstall # or replace master by devel
wstool up                         # now wstool knows which repositories to pull, let's get the
code
cd ~/catkin_ws
catkin build --summary            # ... and finally compile everything
```

3. Indítsuk el a PSM1 (Patient Side Manipulator) RViz szimulációját:

```
roslaunch dvrk_robot dvrk_arm_rviz.launch arm:=PSM1 config:=/home/$(whoami)/
catkin_ws/src/cisst-saw/sawIntuitiveResearchKit/share/console/console-
PSM1_KIN_SIMULATED.json
```

```
---
```

2. PSM subscriber implementálása

1. Nyissuk meg a workspace-t QtCreatorban, mint új ROS workspace.

2. Hozzunk létre új python forrásfájlt `psm_grasp.py` névvel a `~/catkin_ws/src/ros_course/scripts` mappában. Adjuk meg a fájl nevét a `CMakeLists.txt` -ben a megszokott módon.

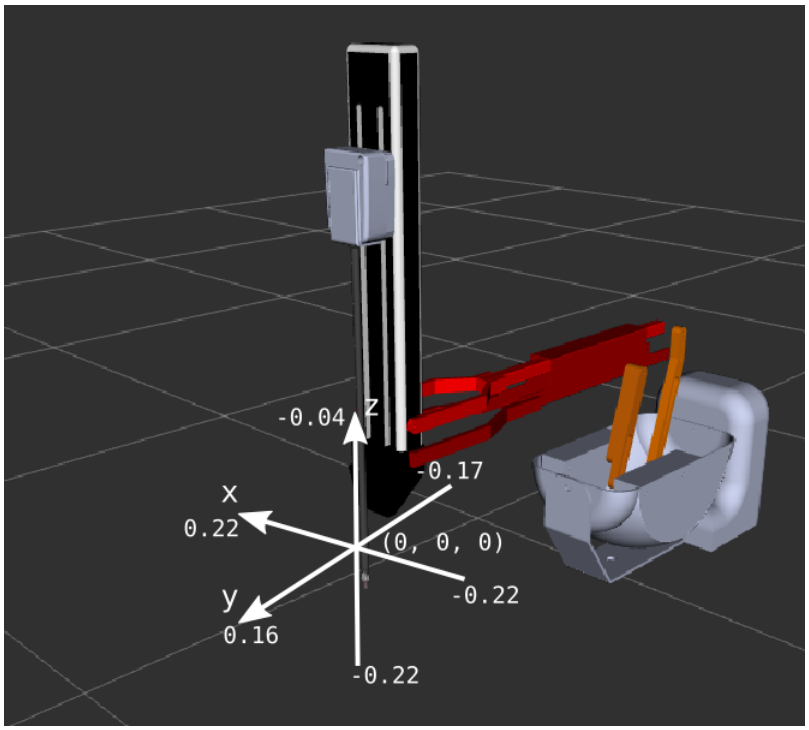
3. Vizsgáljuk a szimulátor működését a tanult prancsok (`rostopic list` , `rosrun rqt_graph rqt_graph` , stb.) használatával. A PSM a lenti topic-okban publikálja a TCP-t (Tool Center Point) és a csipesz pofái által bezárt szöget. Iratkozzunk fel ezekre a topic-okra, írassuk ki és tároljuk el a pillanatnyi állapotot egy-egy változóban.

```
/PSM1/measured_cp
/PSM1/jaw/measured_js
```

4. Build-eljünk és futtassuk a node-ot:

```
cd ~/catkin_ws
catkin build ros_course
rosrun ros_course psm_grasp.py
```

3. PSM TCP mozgatása lineáris trajektória mentén



1. A PSM a lenti topicok-ban várja a kívánt TCP pozíciót és a csipesz pofái által bezárt szöget. Hozzunk létre publishereket a `psm_grasp.py` fájlban ezekhez a topicokhoz.

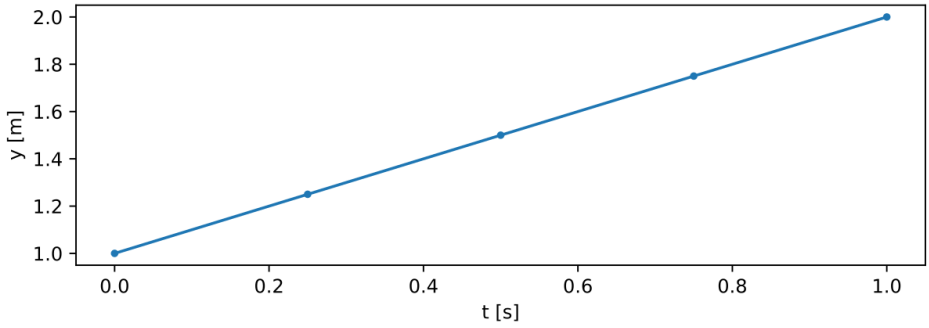
```
/PSM1/servo_cp
/PSM1/jaw/servo_jp
```

2. Írjunk függvényt, amely lineáris trajektória mentén a kívánt pozícióba mozgatja a TCP-t. Küldjük az csipeszt a (0.0, 0.05, -0.12) pozícióba, az orientációt hagyjuk változatlanul. 0.01s legyen a mintavételi idő.

```
def move_tcp_to(self, target, v, dt):
```

3. Írjunk függvényt, amellyel a csipeszt tudjuk nyitni-zárni, szintén lineáris trajektória használatával.

```
def move_jaw_to(self, target, omega, dt):
```



4. Dummy marker létrehozása

1. Hozzunk létre új python forrásfájlt `dummy_marker.py` névvel a `~/catkin_ws/src/ros_course/scripts` mappában. Adjuk meg a fájl nevét a `CMakeLists.txt`-ben a megszokott módon.

2. Vizsgáljuk meg a `visualization_msgs/Marker` msg típust.

3. Implementájunk python programot, amely markert publikál (-0.05, 0.08, -0.12) pozícióval `dummy_target_marker` nevű topic-ban. A `frame_id` addattag értéke legyen `PSM1_psm_base_link`.
4. Futtassuk a node-ot és jelenítsük meg a markert RViz-ben.

5. Marker megfogása

1. Módosítsuk a `psm_grasp.py` programot úgy, hogy a csipesszel fogjuk meg a generált markert.

Note

A használt szimulátor hajlamos rá, hogy bizonyos értékek "beragadjanak", ezért a program elején érdemes az alábbi sorok használatával resetelni a kart:

```
#Reset the arm
psm.move_tcp_to([0.0, 0.0, -0.12], 0.01, 0.01)
psm.move_jaw_to(0.0, 0.1, 0.01)
```

Hasznos linkek

- Download and compile dVRK
- Marker examples
- Numpy vector magnitude
- Numpy linspace