# 06. Robotikai alapfogalmak, da Vinci sebészrobot programozása szimulált környezetben, ROS1-ROS2 bridge

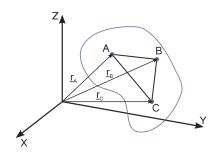
# Elmélet



### **Warning**

**ZH1** (ROS alapok, publisher, subscriber. Python alapok. Launch, paraméterek, rosbag. Robotikai alapfogalmak.) **április 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.

•



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...

# 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]
1[0:2]
1[-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)
1 < 7
l[1 < 7]
np.where (l < \textcolor{red}{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, 24)
C, S = np.cos(X), np.sin(X)

plt.plot(X, C, label='y=cos(x)', marker='.')
plt.plot(X, S label='y=sin(x)', marker='.')

plt.xlabel('x')
plt.ylabel('y')
plt.legend()

plt.show()
```

### If not installed:

```
pip3 install matplotlib
```

# Gyakorlat

Ĭ

# 1: ROS1-ROS2 bridge install

- 1. Nyissuk meg a ~/.bashrc fájlt és kommenteljük ki a ROS 1, ROS 2 és további ROS workspace-ek source-olásáért felelős sorokat.
- 2. Adjuk hozzá az alábbi sort a ~/.bashrc fájlhoz:

```
export ROS_MASTER_URI=http://localhost:11311
```

3. Telepítsük a ros-foxy-ros1-bridge csomagot:

```
sudo apt update
sudo apt install ros-foxy-ros1-bridge
```

### 2: Catkin workspace

1. Telepítsük a catkin build tools csomagot:

```
sudo apt update
sudo apt-get install python3-catkin-tools python3-osrf-pycommon
```

2. Hozzuk létre a catkin workspace-t:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws
catkin init
```

### 3: 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 a ROS verziók source-olását megkönnyítő scriptet (a VM-eken már le van töltve). Source-oljuk a ROS 1-et:

```
source ros_setup.sh -v 1
```

3. 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
```



### Danger

**Soha** ne használjuk a catkin build és a catkin\_make parancsokat ugyanabban a workspace-ben!

4. Indítsuk el a PSM1 (Patient Side Manipulator) RViz szimulációját. A dVRK konzolon ne felejtsünk el HOME-olni. Indítsuk el a ROS1-ROS2 Bridge-t. Tanulmányozzuk a szimulátor működését ROS 2-ből a tanult prancsok (ros2 topic list, ros2 topic echo ros2 run rqt gui rqt gui, stb.) használatával.

```
source ros_setup.sh -v 2
ros2 topic list
ros2 topic echo /PSM1/measured_cp
ros2 run rqt_gui rqt_gui
````rqt_gui` segítségével.

```bash
source ros_setup.sh -v 1
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
```

```bash source ros\_setup.sh -v b ros2 run ros1\_bridge dynamic\_bridge -- bridge-all-topics

```
```bash
source ros_setup.sh -v 2
ros2 run rqt_gui rqt_gui
```

# 4: PSM subscriber implementálása

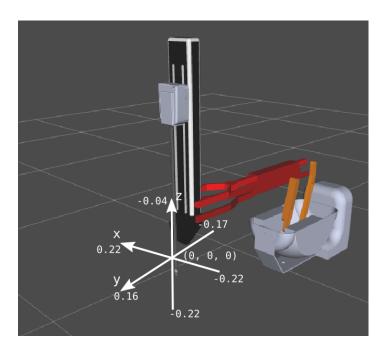
- 1. Hozzunk létre új python forrásfájlt psm\_grasp.py névvel a ~/ros2\_ws/src/ros2\_course/ros2\_course mappában. Adjuk meg az új entry point-ot a setup.py-ban a megszokott módon.
- 2. Iratkozzunk fel a PSM TCP (Tool Center Point) pozícióját és a csipesz pofái által bezárt szögét publikáló topic-okra.

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

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

```
source ros_setup.sh -v 2
cd ~/ros2_ws
colcon build --symlink-install
ros2 run ros2_course psm_grasp
```

5. PSM TCP mozgatása lineáris trajektória mentén



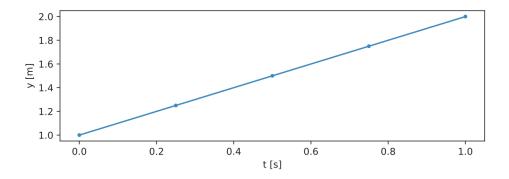
1. A PSM a lenti topicok-ban várja a kívánt TCP pozíciót és a csipesz pofái által bezárt szöget. Hozzunk létre publishereket a psm\_grasp.py fájlban ezekhez a topicokhoz.

```
/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. Matplotlib használatával plotoljuk a tervezett trajektória x, y és z komponensét idő függvényében.

```
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):
```

# 6. Dummy marker létrehozása

1. Hozzunk létre új python forrásfájlt dummy\_marker.py névvel. Adjuk meg az entry point-ot a setup.py -ban a megszokott módon. Implementájunk python programot, amely markert publikál (-0.05, 0.08, -0.14) pozícióval dummy\_target\_marker nevű topic-ban. A frame\_id addattag értéke legyen PSM1\_psm\_base\_link. Másoljuk az alábbi kódot a dummy\_marker.py fájlba:

```
import rclpy
from rclpy.node import Node
from visualization_msgs.msg import Marker

class DummyMarker(Node):
    def __init__(self, position):
        super().__init__('minimal_publisher')
        self.position = position
        self.publisher_ = self.create_publisher(Marker, 'dummy_target_marker', 10)
        timer_period = 0.1 # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
```

```
self.i = 0
    i = 0
  def timer callback(self):
    marker = Marker()
    marker.header.frame id = 'PSM1 psm base link'
    marker.header.stamp = self.get clock().now().to msg()
    marker.ns = "dvrk viz"
    marker.id = self.i
    marker.type = Marker.SPHERE
    marker.action = Marker.MODIFY
    marker.pose.position.x = self.position[0]
    marker.pose.position.y = self.position[1]
    marker.pose.position.z = self.position[2]
    marker.pose.orientation.x = 0.0
    marker.pose.orientation.y = 0.0
    marker.pose.orientation.z = 0.0
    marker.pose.orientation.w = 1.0
    marker.scale.x = 0.008
    marker.scale.y = 0.008
    marker.scale.z = 0.008
    marker.color.a = 1.0 \# Don't forget to set the alpha!
    marker.color.r = 0.0
    marker.color.g = 1.0
    marker.color.b = 0.0;
    self.publisher_.publish(marker)
    self.i += 1
def main(args=None):
  rclpy.init(args=args)
  marker_publisher = DummyMarker([-0.05, 0.08, -0.12])
  rclpy.spin(marker publisher)
  # Destroy the node explicitly
  # (optional - otherwise it will be done automatically
  # when the garbage collector destroys the node object)
  marker publisher.destroy node()
  rclpy.shutdown()
if name == ' main ':
  main()
```

2. Futtassuk a node-ot és jelenítsük meg a markert RViz-ben.

# 7. Marker megfogása

1. Iratkozzunk fel a marker pozícióját küldő topic-ra a psm grasp.py -ban.

2. 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:

# Hasznos linkek

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