



## 07. Kinematics, inverse kinematics. Programming a simulated robot in joint space and task space

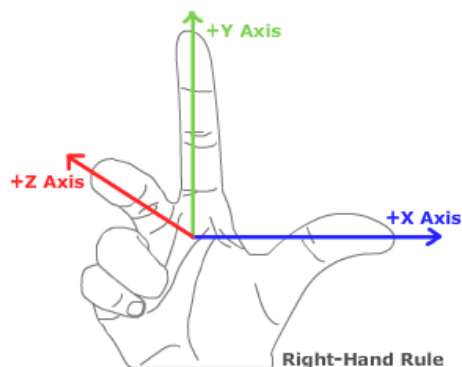
### Warning

**ZH2** (Roslaunch, ROS parameter server, ROS service, ROS action, Kinematics, inverse kinematics) and **project presentation: December 6.**

## Rehearsal

### 3D transformations

•



**Position:** 3D offset vector

- **Orientation:** 3 x 3 rotation matrix
  - further orientation representations: Euler-angles, RPY, angle axis, quaternion
- **Pose:** 4 x 4 (homogenous) transformation matrix
- **Frame:** origin, 3 axes, 3 base vectors, right hand rule
- **Homogenous transformation:** rotation and translation in one transformation
  - e.g., for the rotation  $\mathbf{R}$  and translation  $\mathbf{v}$ :

$$\begin{aligned} \mathbf{T} &= \begin{bmatrix} \mathbf{R} & \mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix} \\ &= \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & v_x \\ r_{2,1} & r_{2,2} & r_{2,3} & v_y \\ r_{3,1} & r_{3,2} & r_{3,3} & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

- **Homogenous coordinates:**

- **Vector:** extended with 0,  $\mathbf{a}_H = \begin{bmatrix} \mathbf{a} \\ 0 \end{bmatrix} = \begin{bmatrix} a_x \\ a_y \\ a_z \\ 0 \end{bmatrix}$
- **Point:** extended by 1,  $\mathbf{p}_H = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}$
- Applying transformations is much easier:

$$\mathbf{q} = \mathbf{R}\mathbf{p} + \mathbf{v} \rightarrow \begin{bmatrix} \mathbf{q} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{v} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$$

- **Degrees of Freedom (DoF):** the number of independent parameters.

## Principles of robotics



- Robots are built of: **segments** (or links) és **joints**
- **Task space** (or cartesian space):
  - 3D space around us, where the task, endpoint trajectories, obstacles are defined.

- **TCP** (Tool Center Point): Frame fixed to the end effector of the robot.
- **Base frame, world frame**
- **Joint space:**
  - Properties or values regarding the joints.
  - Low-level controller.
  - Joint angles, joint velocities, accelerations, torques....

## Lecture

### Kinematics, inverse kinematics

#### Def. Kinematics

Calculation of the pose of the TCP from joint angles. (From joint space to task space)

- Kinematic model
  - Denavit--Hartenberg (HD) convention
  - URDF (Unified Robotics Description Format, XML-based)

If the frames attached to each segment are named  $(base, 1, 2, 3, \dots, TCP)$ , transformations between two neighboring segments  $(i)$  and  $(i+1)$ ---dependent on the angle of the joint enclosed by them---are named  $(T_{i+1,i}(q_{i+1}))$ , the transformation from the base frame to the TCP can be calculated as follows for a robot with  $(n)$  joints:

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

#### Def. Inverse kinematics

Calculation of the joint angles in order to reach desired (or any) TCP pose. (From task space to joint space)

## Differential inverse kinematics

### Def. Differential inverse kinematics

How to change the joint angles to achieve the desired **small** change in TCP pose (including rotation and translation).

- **Jacobian matrix** (Jacobian): The Jacobian matrix of a vector-valued function of several variables is the matrix of all its first-order partial derivatives.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x_1}{\partial q_1} & \frac{\partial x_1}{\partial q_2} & \frac{\partial x_1}{\partial q_3} & \dots & \frac{\partial x_1}{\partial q_n} \\ \frac{\partial x_2}{\partial q_1} & \frac{\partial x_2}{\partial q_2} & \frac{\partial x_2}{\partial q_3} & \dots & \frac{\partial x_2}{\partial q_n} \\ \frac{\partial x_3}{\partial q_1} & \frac{\partial x_3}{\partial q_2} & \frac{\partial x_3}{\partial q_3} & \dots & \frac{\partial x_3}{\partial q_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_m}{\partial q_1} & \frac{\partial x_m}{\partial q_2} & \frac{\partial x_m}{\partial q_3} & \dots & \frac{\partial x_m}{\partial q_n} \end{bmatrix}$$

- **Jacobian matrix in robotics**: defines the relationship between the joint velocities and the velocity of the TCP:

$$\begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix} = \mathbf{J}(\mathbf{q}) \dot{\mathbf{q}}$$

If  $\Delta t$  is small enough:

$$\begin{aligned} & \frac{\Delta \mathbf{r}}{\Delta t} \approx \mathbf{J}(\mathbf{q}) \frac{\Delta \mathbf{q}}{\Delta t} \end{aligned}$$

## Differential inverse kinematics using Jacobian inverse

1. Calculate the difference of the desired and the current position:  $\Delta \mathbf{r} = \mathbf{r}_{\text{desired}} - \mathbf{r}_0$
2. Calculate the rotation between the current orientation and the desired orientation:  $\Delta \mathbf{R} = \mathbf{R}_{\text{desired}} \mathbf{R}_0^T$ , majd konvertáljuk át axis angle reprezentációba  $(\mathbf{t}, \phi)$
3. Calculate  $\Delta \mathbf{q} = \mathbf{J}^{-1}(\mathbf{q}_0) \begin{bmatrix} \mathbf{k}_1 \Delta \mathbf{r} \\ \mathbf{k}_2 \Delta \phi \end{bmatrix}$

$\right]$ ), where the inverse could be substituted by pseudo-inverse or transpose

4. Change joint angles:  $\mathbf{q}_{\text{better}} = \mathbf{q}_0 + \Delta \mathbf{q}$

## Practice

### 1: Install rrr-arm

1. Install dependencies:

```
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
```

#### Tip

We will use the package `kinpy` to calculate forward kinematics. Download the source of the package `kinpy` and study the API: <https://pypi.org/project/kinpy/>

2. Clone and build the repo:

```
cd ~/catkin_ws/src
git clone https://github.com/Robotawi/rrr-arm.git
cd ..
catkin build
```

3. Launch the simulator and move the arm:

```
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

The simulator might raise errors like "No p gain specified for pid...", but those can be ignored as won't cause any issues.

## 4. Build the URDF file describes the robot:

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

## 2: Move the arm in joint space

1. Create a new file named `rrr_arm_node` in the `scripts` folder. Add it to the `CMakeLists.txt`, as usual. Subscribe to the topic which publishes the joint angles (configuration) of the robot. Create publishers to the 4 topics setting the joint angles of the arm. Use the previous Python scripts as a template.

### Warning

Gazebo and `kinpy` organized the joints in different orders: **1. [gripper\_joint\_1, gripper\_joint\_2, joint\_1, joint\_2, joint\_3, joint\_4]** - topic `/rrr_arm/joint_states` - method `kp.jacobian.calc_jacobian(...)`

**2. [joint\_1, joint\_2, joint\_3, joint\_4, gripper\_joint\_1, gripper\_joint\_2]** - method `chain.forward_kinematics(...)` - method `chain.inverse_kinematics(...)`

2. Send the arm to the configuration [1.0, 1.0, 1.5, 1.5].

### 3. Kinematic task

1. Import `kinpy` and read the URDF of the robot:

```
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. Calculate the TCP pose in the current configuration using `kinpy`. The example at <https://pypi.org/project/kinpy/> is wrong, use the following example:

```
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: Inverse kinematics using Jacobian inverse

Implement a method calculating inverse kinematics using Jacobian inverse for the robot. Ignore the orientation for now. Move the TCP to position `(0.59840159, -0.21191189, 0.42244937)`.

1. Write a while loop stopping if the length of `delta_r` is below threshold or `rospy.is_shutdown()`.
2. Calculate the difference of the desired and current TCP positions (`delta_r`). Scale by constant `k_1`.
3. Let `phi_dot_t` be `[0.0, 0.0, 0.0]` (ignore the orientation).
4. Concatenate `delta_r` and `phi_dot_t`.



5. Calculate the Jacobian matrix in the current configuration using the method `kp.jacobian.calc_jacobian(...)`.
6. Calculate the pseudo inverse of the Jacobian using `np.linalg.pinv(...)`.
7. Calculate `delta_q`, use the `.dot(...)` method from Numpy.
8. Increase the joint angles by `delta_q`.

## Bonus exercise: Inverse kinematics with orientation

Extend the previous exercise by calculating both the TCP position and orientation.

## Useful links

- [rrr-arm model](#)
- <https://pypi.org/project/kinpy/>
- [https://en.wikipedia.org/wiki/Axis%E2%80%93angle\\_representation](https://en.wikipedia.org/wiki/Axis%E2%80%93angle_representation)
- <https://www.rosroboticslearning.com/jacobian>