Introduction to Robotics

LAB 3 – CONTROL OF A NEW ROBOT PERFORMING A SPECIFIC TASK

Objectives

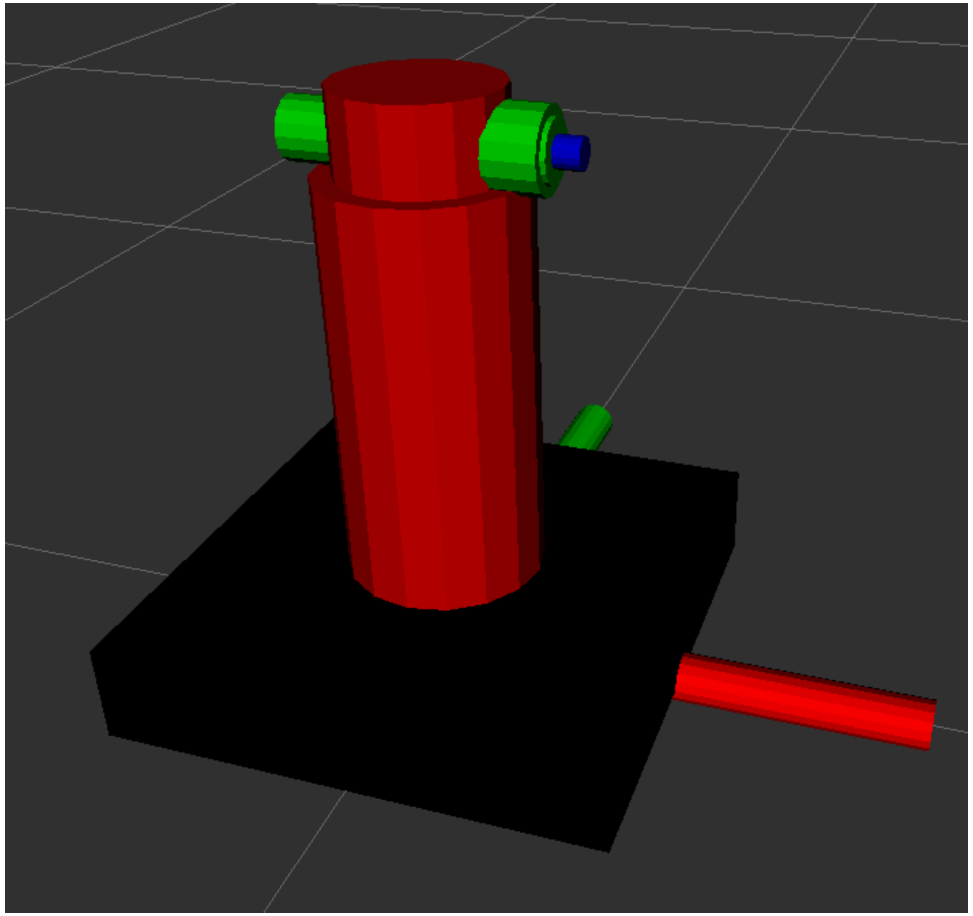
Objectives

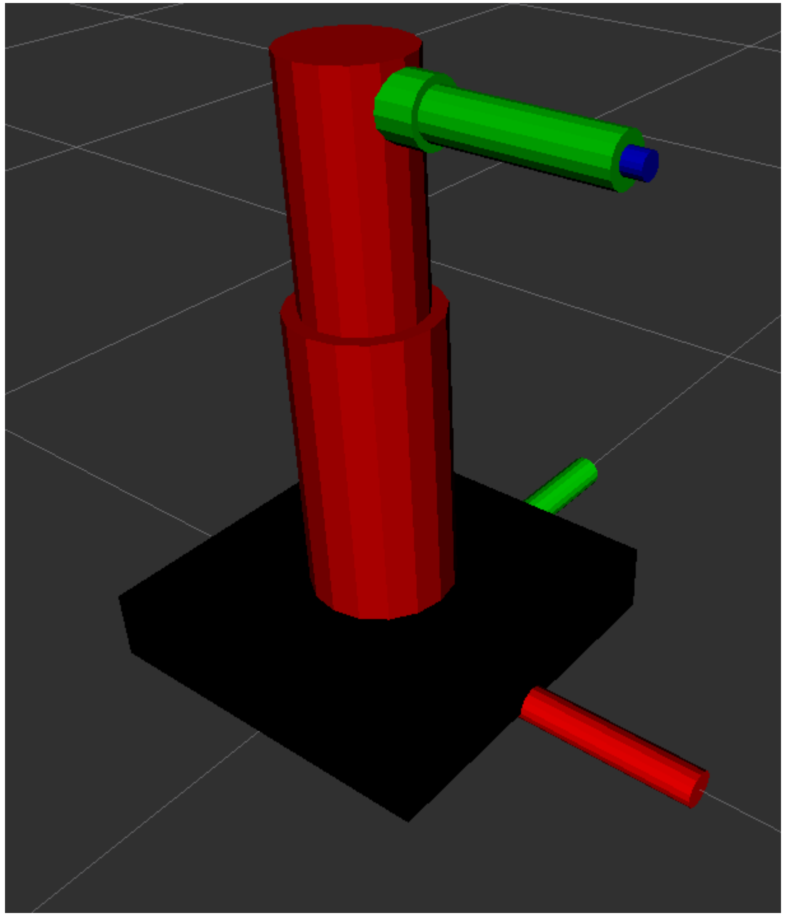
Apply the methodology described in Lab 2 to control a new robotic manipulator for a specific task.

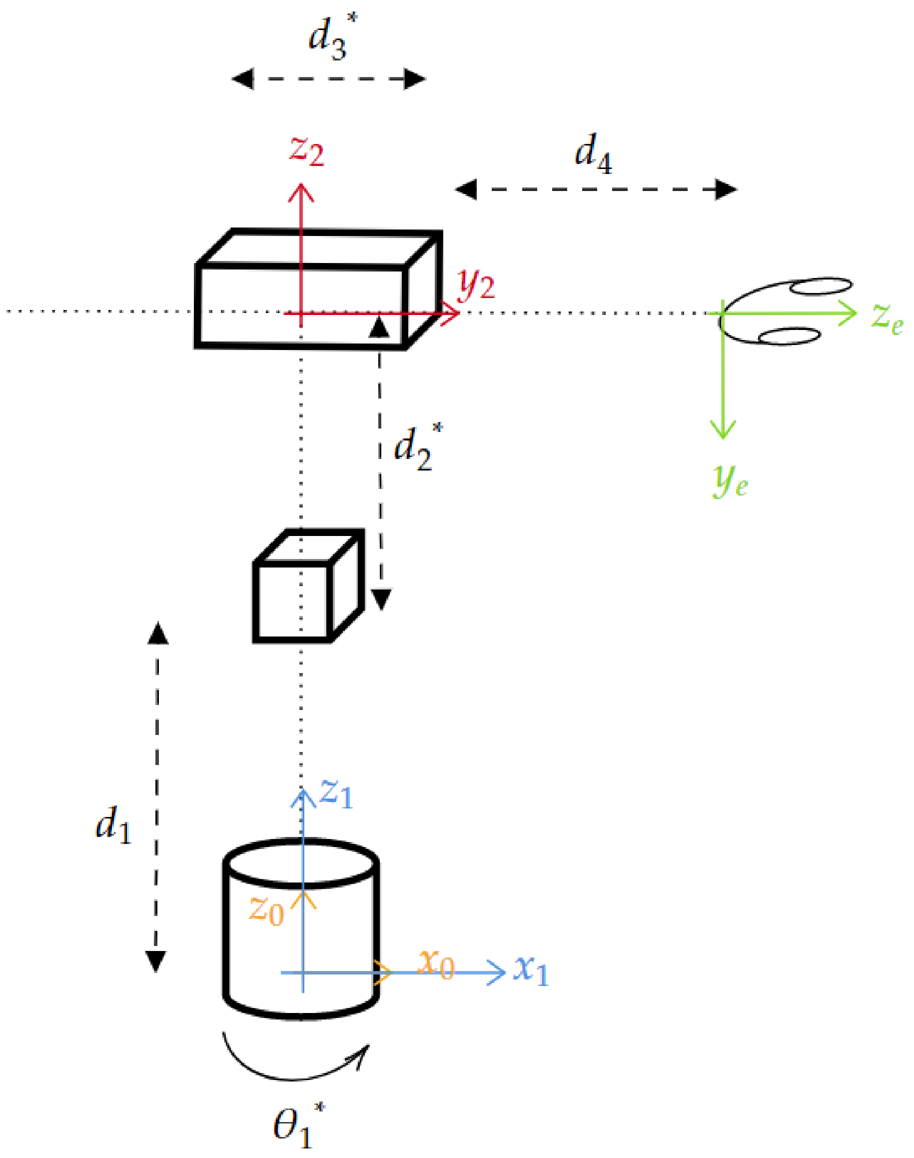
- Determine the FK of the robot.
- Compute its IK.
- Define a set of actions the robot will perform for a specific task.

Part 1
Analysis of a new robot

3D representation of the robot







2D representation of the robot

$\mathcal{R}_0$: base link (fixed)
$\mathcal{R}_1$: link 1
$\mathcal{R}_2$: link 2
$\mathcal{R}_e$: End effector link

Exercises

Referring to the previous scheme, determine the modified Denavit-Hartenberg parameters

$S_{i\to i+1}$	a_i	α_i	d_{i+1}	θ_{i+1}
$S_{w\to 0}$	0	0	d_0	0
$S_{0\to 1}$				
$S_{1\to 2}$				
$S_{2\to e}$				

Parameters	d_0	d_1	d_4
Values (meter)	0.05	0.48	0.15

Exercises

Based on the previous table, determine the homogeneous transformation matrix for each joint:

$$T_{i\rightarrow i+1} = T_{i+1}^{i} = \begin{bmatrix} c_{\theta_{i+1}} & -s_{\theta_{i+1}} & 0 & a_i \\ s_{\theta_{i+1}} c_{\alpha_i} & c_{\theta_{i+1}} c_{\alpha_i} & -s_{\alpha_i} & -s_{\alpha_i} d_{i+1} \\ s_{\theta_{i+1}} s_{\alpha_i} & c_{\theta_{i+1}} s_{\alpha_i} & c_{\alpha_i} & c_{\alpha_i} d_{i+1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Exercises

1. Deduce the forward kinematics

2. Determine the coordinates (x, y, z) of the end effector for each configuration (θ_1, d_2, d_3):

$(\boldsymbol{\theta_1}, \boldsymbol{d_2}, \boldsymbol{d_3})$	**x**	**y**	**z**
$(0, 0, 0)$			
$\left(\frac{\pi}{2}, 0.1, 0.05\right)$			
$\left(-\frac{\pi}{4}, 0.2, 0.13\right)$			

Exercises

1. Open your project on ROS Development Studio

2. Run the following instruction to see the robot on RViz

   ros2 launch in426_simu display_2_launch.py

3. To compare your results (previous table) with the coordinates of the end effector displayed on RViz, modify manually the joints with the joint_state_publisher GUI.

Exercises

1. Based on the equations given by the forward kinematics, determine the inverse kinematics.

2. Complete the following table using the IK:

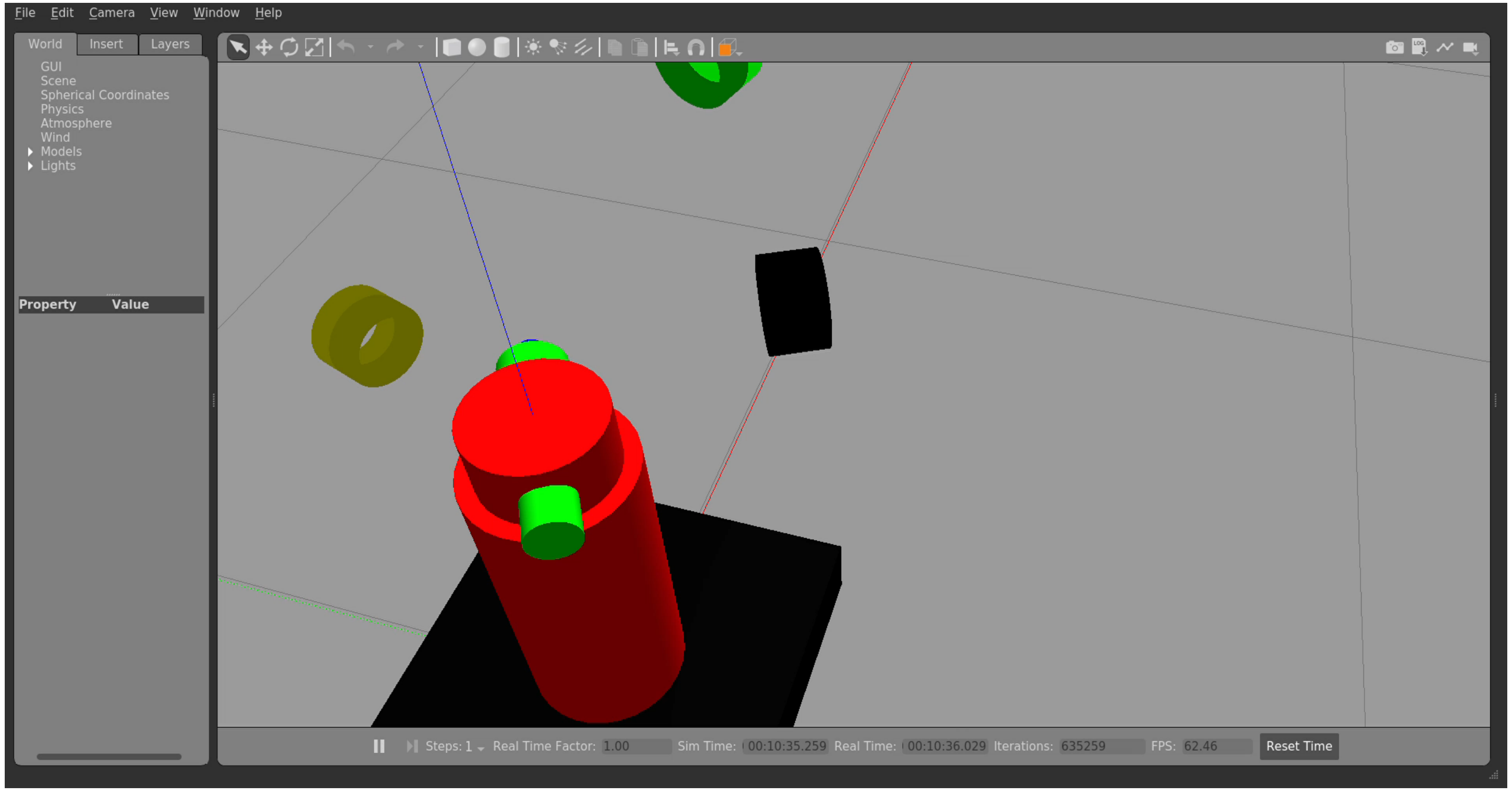
(x, y, z)	θ_1	d_2	d_3
$(0.2, 0.2, 0.6)$			
$(0.13, -0.26, 0.7)$			
$(0, -0.2, 0.55)$			

3. To compare your results (previous table) with the coordinates displayed on RViz, run the following instruction, then modify manually the joints

ros2 launch in426_simu display_2_launch.py

Part 2
Robot performing a task

Instructions



This video is available on Moodle

Implementation of the motion

READ CAREFULLY THE FILE **MOTION_ROBOT_2.PY** AND EDIT ITS METHODS **IK()** AND **EXECUTE()**

```
+---in426_motion
|    |    package.xml
|    |    setup.cfg
|    |    setup.py
|    |
|    +---in426_motion
|    |         ik_1.py
|    |         motion_robot_2.py
|    |         send_goal_test.py
|    |         __init__.py
|    |
|    +---launch
|    |         ik_1_launch.py
|    |         motion_robot_2_launch.py
```

Implementation of the IK

Based on the inverse kinematic of the robot (obtained at part 1), complete the method ***ik()***. You have to use the existing class attributes.

Defining the waypoints

Complete the method ***execute()*** as follow:

- For each desired position provided in the comment section:
  - Determine the corresponding joint angles using the ***ik()*** method.
  - Add a ***JointTrajectoryPoint()*** message to the list ***points***. It corresponds to a waypoint. You can specify the moment this point should be reached.

- The list of waypoints will be sent as a ROS2 action goal at the end of this method.

- Refer to next slide to see an example.

Defining the waypoints

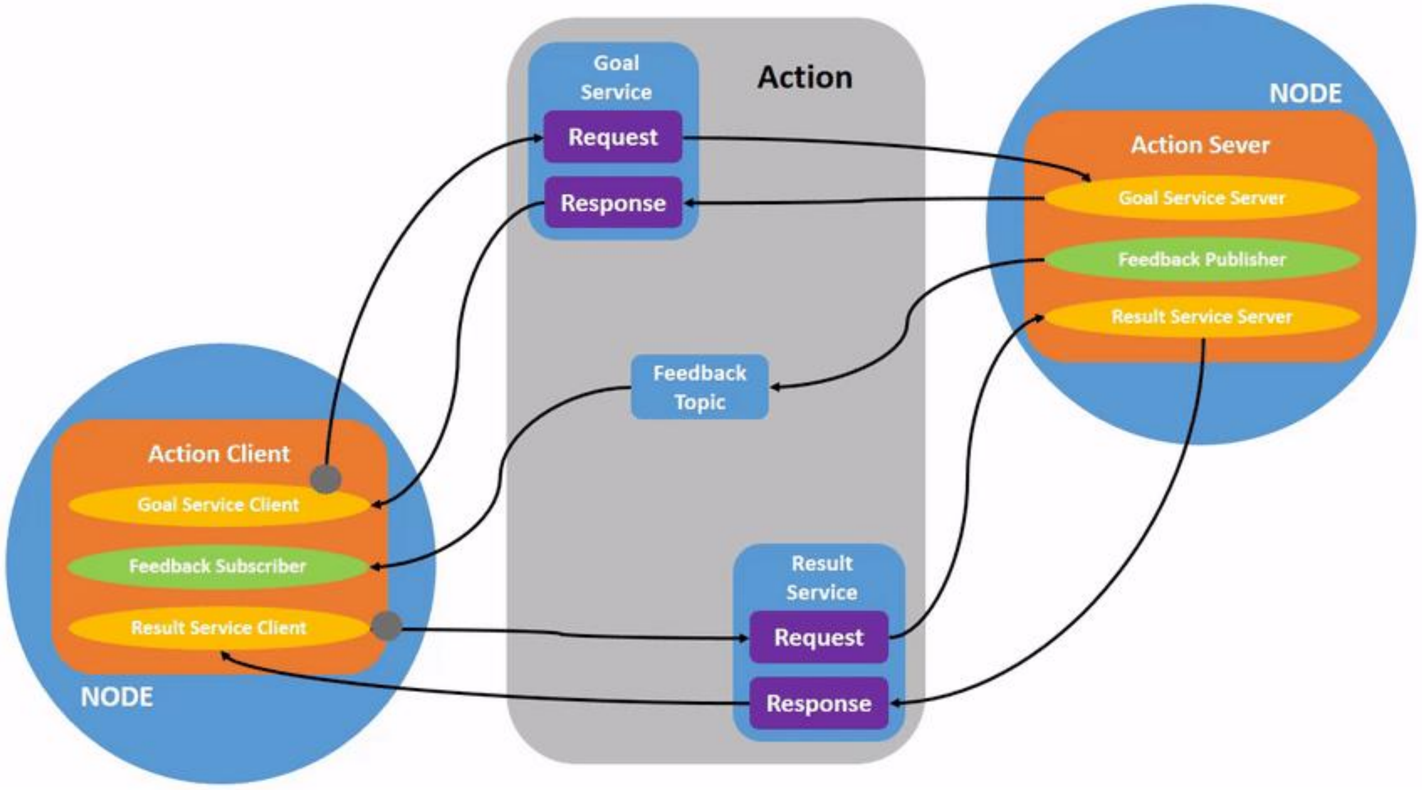
```python
points = []

#Move to coordinates x=1, y=2, z=3
self.ik(1, 2, 3)    #self.q1, self.q2 and self.q3 are automatically updated
point1 = JointTrajectoryPoint()
point1.time_from_start = Duration(seconds=2, nanoseconds=0).to_msg()
point1.positions = [self.q1, self.q2, self.q3]
points.append(point1)

#Move to coordinates x=4, y=5, z=6
self.ik(4, 5, 6)    #self.q1, self.q2 and self.q3 are automatically updated
point2 = JointTrajectoryPoint()
point2.time_from_start = Duration(seconds=7, nanoseconds=0).to_msg()
point2.positions = [self.q1, self.q2, self.q3]
points.append(point2)

self.send_joints(points)
```

Let's assume the robot has to reach two waypoints with coordinates $(1,2,3)^T$ and $(4,5,6)^T$.

The first waypoint should be reached after 2s while the second should be reached 5s later.

Note that the duration for point2 is 7s not 5s because it corresponds to the time elapsed since the robot started its mission.

Introduction to ROS2 actions



This animation comes from the [official ROS2 Galactic documentation]

Run the robotic application

1. Launch the simulation (terminal 1):

   ros2 launch in426_simu simu_2_launch.py

2. Wait for the simulation to start, then run the task manager node (terminal 2):

   ros2 launch in426_motion motion_robot_2_launch.py

3. You should see the robot perform the given task. If the robot does not move as expected, modify your solution in the python script and execute again steps 1 and 2.

Submission

Content and deadline

- You must submit on Moodle **after Lab 3** the following elements:
  - A report (PDF format in english)
  - Your implementations (ik_1.py and motion_robot_2.py)

- Your report must contain:
  - Introduction
  - Answers to all the questions of **Labs 2 and 3**
  - **ALL** the steps that helped you obtain the forward and inverse kinematics of **both** robotic arms. In your equations, keep the variables not their values (e.g: you write L_1, not 0.43)
  - Conclusion

- **One submission per group**