Advanced Robotics

Lab 3 – ROS / Kinematics

Pôle

Numérique

Author	: Guillaume GIBERT
Version	: 1 R 0
Promotion	: 2026
Program	: EENG4
Date	: December 10th 2024

1. Lab description

The aim of this lab session is to control a robot arm and to experiment with direct and inverse kinematics. The students will experiment using a custom version of the Poppy Ergo Jr robot arm. By the end of the session, the robot should grasp soft coloured cubes and move them to form a tower. This lab session is composed of different phases:

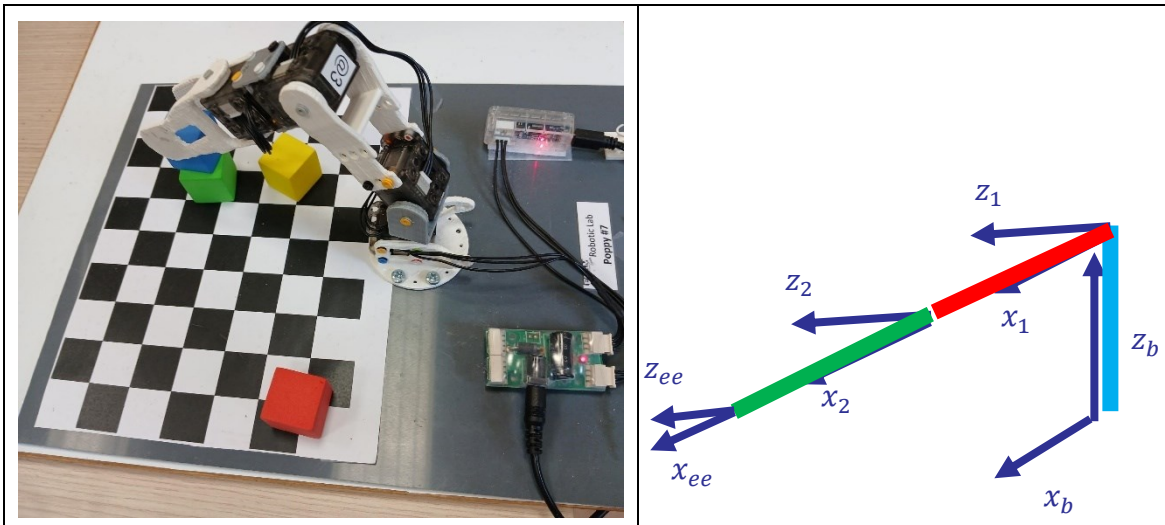
- To set up the hardware components;
- To launch the provided ROS components;
- To create a ROS node that drives the robot to move cubes and create a tower.

As input, the students will receive a "Methods" section of a scientific article describing the hardware and software setup and the corresponding code. Based on this code, they will run the testing procedure and obtain some results that will be reported in the "Results" section of the article. Then, they will develop a ROS node to drive the robot to build towers of soft cubes based on their Cartesian positions. This application should be described in the "Methods" section and the obtained results should be reported in the "Results" section.

2. Methods

2.1. Setup

One custom Poppy Ergo Jr robot arm is used in this experiment. The original Poppy Ergo Jr robot [1] is composed of seven revolute joints. Our custom version is only composed of four revolute joints (see Figure 1a). Each joint is driven by a Dynamixel XL-320 servomotor [2].



(a) Custom Poppy Ergo Jr robot

(b) Reference frames of the robot

Figure 1 : Custom Poppy Ergo Jr robot composed of four revolute joint (a) picture, (b) schematic representation of the reference frames of the robot.

Each joint is connected to the next one thanks to 3D printed elements. In order to perform action with a robotic system, a power supply and a controller are required. The power supply (7.5V 2A) provides the necessary power to the servo motors. To this end, the voltage and the current of the power supply should meet the servo motor needs. The power converter is connected to the SMPS2Dynamixel power board [3]. The controller is either a microcontroller or a computer that can send commands to the servo motors and also receive the robot state parameters (e.g., current position, torque...). In our setup, the U2D2 control board is used [4]. It is a USB communication converter that enables to control and operate Dynamixel servo motors with a PC. The architecture of the robotic system of this lab is represented in Figure 2.

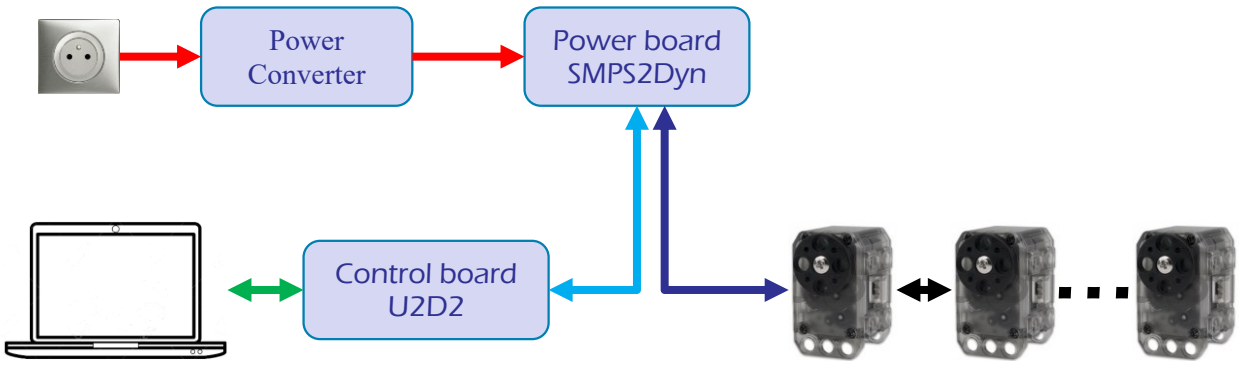


Figure 2 : Schematic representation of the robotic system architecture. A Power board provides sufficient voltage and current to the servo motors. A Control board converts the commands sent by a computer into TTL signal that servo motors can interpret. It also converts the robot state parameters (from TTL) into values that the PC can interpret.

2.2. Kinematics

The robot is composed of 4 revolute joints: the first three ones move the end-effector in space whereas the last one actuate the gripper. Therefore, this robot has three degrees of freedom to move in 3D space. It can move its end-effector to any position in its working space, but it cannot impose a given orientation. The reference frames of the different joints are represented in Figure 1b and the Denavit-Hartenberg parameters of this robot are provided in Table 1.

Table 1 : Denavit-Hartenberg parameters of the custom Poppy Ergo Jr robot

Joint i	Θ	d	a	α
1	q1	L0	0	90°
2	q2	0	L1	0
3	q3	0	L2	0

The Forward Kinematics for the position only of the end-effector is

$$\begin{cases} x = (L_2 \sin(q_2) + L_3 \sin(q_2 + q_3)) \cos(q_1) \\ y = (L_2 \sin(q_2) + L_3 \sin(q_2 + q_3)) \sin(q_1) \\ z = L_2 \cos(q_2) + L_3 \cos(q_2 + q_3) + L_1 \end{cases}$$

The Inverse Kinematics for the position only of the end-effector is:

$$\begin{cases} q_1 = \operatorname{atan2}(y, x) \\ q_2 = atan2\left(\sqrt{x^2 + y^2}, z - L_1\right) - atan2(L_2 \sin(q_3), L_1 + L_2 \cos(q_3)) \\ q_3 = \pm \operatorname{acos}\left(\dfrac{(z - L_1)^2 + (x^2 + y^2) - {L_2}^2 - {L_3}^2}{2 L_2 L_3}\right) \end{cases}$$

2.3. ROS settings

The custom Poppy Ergo Jr robot arm is physically connected through USB to a laptop under Ubuntu 20.04. The ROS environment Noetic [5]is installed on the laptop. ROS provides libraries and tools to develop software for robotic applications. Each laptop will run locally (i.e. there is no need to be connected to a network) a ROS Master as represented in Figure 3. Therefore, the ROS environment variables were set as follow in the ~/.bashrc file:

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

```
ROS_HOSTNAME=localhost
```

The ROS master was launched on the PC and provided naming and registration services to all the nodes in the ROS environment.

A package entitled advrobotics_lab3 was created with ros_cpp and std_msgs as dependencies.

2.3.1. Message types

Two types of custom messages were created using the procedure described in [6]:

- **joints** composed of three float32 values q1, q2 and q3;
- **gripper** composed of one float32 value gripper.

A custom service message was also created using the procedure described in [6]:

- **ikpoppy** composed of
  - three float32 values for the request: x, y and z;
  - four float32 values for the response: sol, q1, q2 and q3.

2.3.2. Nodes

The node poppy_core retrieves the current joint values thanks to the Dynamixel Handler class at a frequency of 10 Hz. Then, it publishes (at the same frequency) these current joint values to the topics:

- /joint_position using the message type **joints**
- /gripper_position using the message type **gripper**.

This node is subscribing to the topics:

- /joint_cmd expecting target joint values to drive the robot motors and therefore to move the end-effector (using the message type **joints**);
- /gripper_cmd expecting angle value to open/close the gripper (using the message type **gripper)**.

As represented in Figure 3, the node ik_server creates a service ik and waits for a request (**ikpoppy** service message): a Cartesian position of the end-effector. As soon as the request arrives, it calls the inverseKinematics() method of the Kinematics object. This method returns the estimated joint values that corresponds to the target Cartesian position of the end-effector. Then, the node should return these joint values to the service ik.

The node ik_client sends a request (i.e. Cartesian position of the end-effector) to the service ik. The node ik_server sends back the estimated joint values. As soon as the node ik_client receives the joint values, it sends the target joint values to the topic /joint_cmd and the robot moves accordingly.

The code is available at this address: https://git.ecam.fr/AdvRobotics.EENG4/advrobotics_lab3.git under the GNU GPL v3 license.

2.4. Testing procedure

Once the poppy_core node is launched; it should create 4 topics:

- /joint_cmd;
- /gripper_cmd;
- /joint_position;
- /gripper_position.

The first step to test our system is to verify that these topics were created using the following command:

```
rostopic list
```

The second step consists of connecting the rqt_plot tool to the following signals:

- /joint_position/q1
- /joint_position/q2

- /joint_position/q3
- /gripper_position_gripper

And to publish commands on the /joint_cmd and /gripper_cmd topics using the following commands:

```
rostopic pub /joint_cmd advrobotics_lab3/joints '{q1: 10.0, q2: -20.0, q3: -15.0}'

rostopic pub /joint_cmd advrobotics_lab3/gripper '{gripper: -40.0}'
```

The robot and the gripper should move and the rqt_plot should display the evolution over time of the joint values.

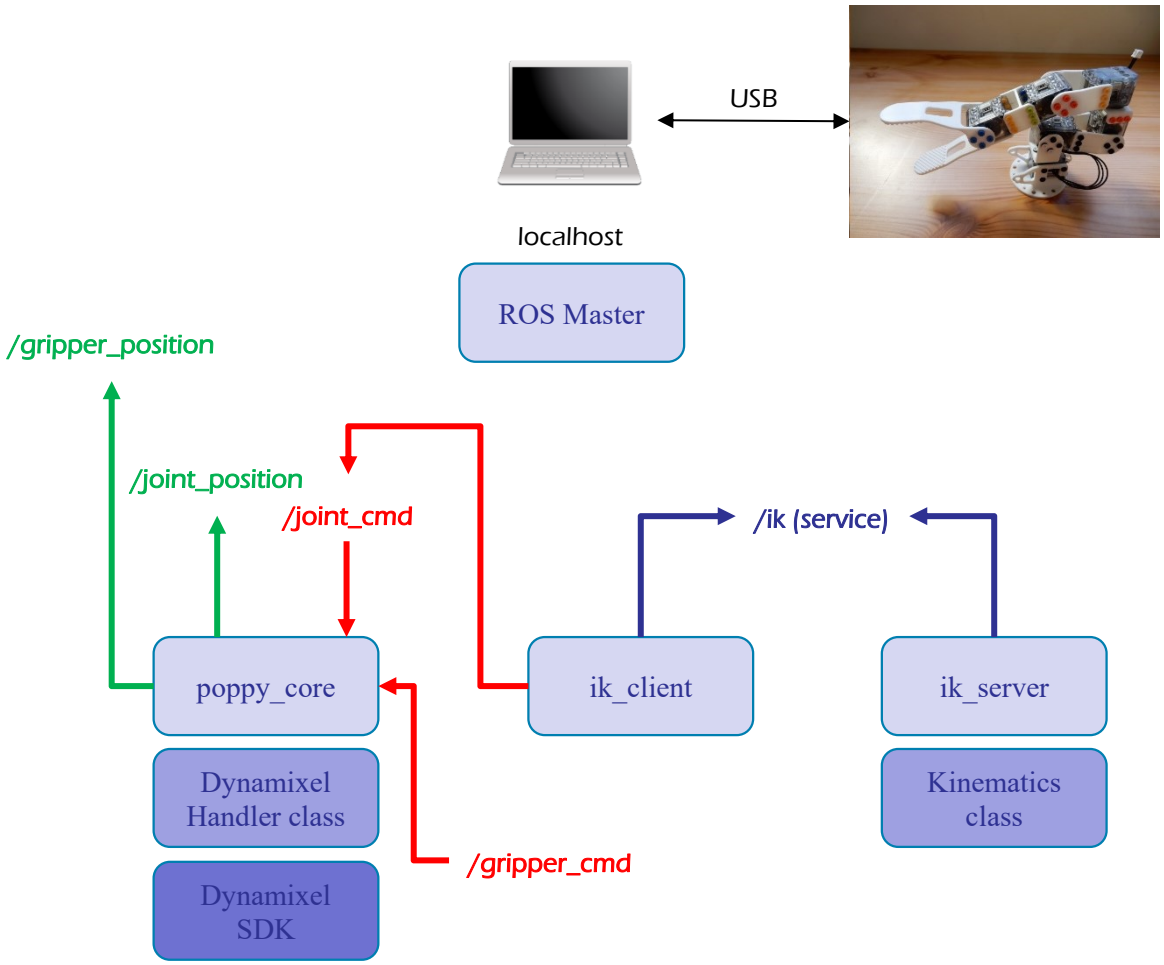


Figure 3 : Schematic representation of the ROS architecture: the node *poppy_core* publishes the current joint values to the topic /joint_position and the gripper value to the topic /gripper_position. It subscribes to the topic /joint_cmd (waiting for target joint values) and also to the /gripper_cmd (waiting for gripper joint value); the node *ik_server* creates a service and as soon as a request comes from the node ik_client it delivers the joint values corresponding to the target Cartesian position and the node ik_client publishes the target joint values to the /joint_cmd topic.

The final testing procedure corresponds to the ik service: the ik_server node is launched in a terminal and then the ik_client is launched in a separate terminal with destination coordinates as arguments as follow:

```
rosrun advrobotics_lab3 ik_client 6 9 0
```

2.5. Application: building a tower

TO BE DONE

3. Tasks

- To setup the system
- To run the testing procedure and report the results
- To develop the node driving the robot to build a tower of soft cubes
- To report the developed node in the "Methods" section
- To describe the obtained results in the "Results" section

4. Report Instructions

- You must write a report that contains a method section and a result section.
- You will be assessed on the final report and the corresponding source code (i.e. provide the link to your git repository).
- You should use the provided Word template to write your report.
- You must submit the report written in **English** as a **pdf** file through Moodle before Wednesday December 11th 2024 for Group C and December 18th 2024 for Group D.
- The title of the report should be **AdvancedRobotics_Lab3_*FirstStudentName_SecondStudentName*.pdf** (Change *FirstStudentName* and *SecondStudentName* accordingly).

5. References

[1] INRIA Flowers lab, «https://www.poppy-project.org/en/robots/poppy-ergo-jr/,» [En ligne].

[2] Dynamixel, «https://emanual.robotis.com/docs/en/dxl/x/xl320/,» [En ligne].

[3] Dynamixel, «http://www.robotis.fr/index.php?id_product=108&controller=product,» [En ligne].

[4] Dynamixel, «https://emanual.robotis.com/docs/en/parts/interface/u2d2/,» [En ligne].

[5] ROS, «http://wiki.ros.org/noetic,» [En ligne].

[6] ROS, «http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv#Creating_a_srv,» [En ligne].