ROBT 611 - Final Project: Snake robot on ROS2

Madina Turmagambetova and Kanagat Kantoreyeva

***Abstract*—The evolution from ROS (Robot Operating System) to ROS2 represents a significant shift in the robotics ecosystem, providing improved performance, scalability, and security to meet modern industry standards. This project aims to transfer the existing laboratory sessions for the course “Robotics II: Control, Modeling and Learning” from ROS to ROS2, ensuring that students gain hands-on experience with contemporary tools essential for robotic system design and implementation. By evaluating the feasibility of transitioning current lab modules and adapting the necessary packages, the project focuses on the migration of the first three labs on a real snake robot. Challenges such as hardware overheating issues, firmware updates, and position control will be addressed. The project will rely on the refactoring of existing ROS packages, the adaptation of nodes, and the modification of custom message definitions to ensure compatibility with ROS2 standards. This transition not only modernizes the course curriculum, but also equips students with relevant expertise for real-world robotic applications.**

I. Introduction

The Robot Operating System (ROS) has long been a cornerstone of robotics research and application, offering a modular and flexible framework that has empowered innovation across industries such as automation, healthcare, and manufacturing. However, with the release of ROS2, the field has witnessed a paradigm shift, introducing features such as enhanced performance, robust scalability, and improved security. These advancements align with modern industrial demands, making ROS2 an essential skill set for robotics professionals. Transitioning from ROS to ROS2 is imperative to ensure students and practitioners remain adept with cutting-edge technologies.

This project focuses on transferring laboratory sessions from the course “Robotics II: Control, Modeling, and Learning” to the ROS2 ecosystem. The current labs introduce students to essential robotics concepts such as ROS architecture, Gazebo simulation, RVIZ visualization, trajectory planning, and inverse kinematics using MoveIt. These sessions involve the control of a real snake robot, providing a comprehensive learning experience. Migrating these labs to ROS2 offers students the opportunity to engage with the latest tools, reflecting industry trends, and ensuring their readiness for advanced challenges in robotics.

The transition process, however, presents challenges, including incompatibility of existing ROS packages, differences in communication protocols, and hardware issues. By leveraging previously developed ROS packages and adapting them to the ROS2 framework, this project aims to overcome these hurdles. The migration will involve refactoring codebases, modifying URDF models, and ensuring compatibility with ROS2’s middleware, message-passing systems, and simulation tools. Successfully transferring these labs to ROS2 will not only modernize the course but also provide students with hands-on experience with state-of-the-art robotic tools and methodologies, preparing them for future innovations in the field.

II. Installation

A. Install Dynamixel Wizard 2.0

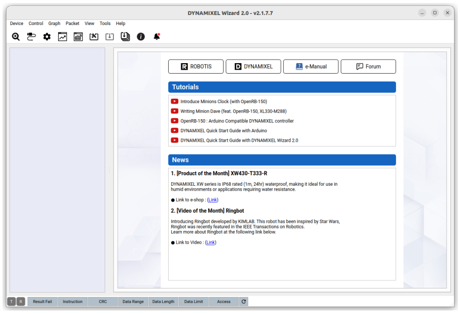
This installation was performed on a Nitro 5 laptop with a dual-boot setup of Windows and Ubuntu 22.04 in November 2024. The Dynamixel Wizard was used for the calibration and ID assignments of the Dynamixel motors.

Note: Although the official supported operating systems for the Dynamixel Wizard are listed as up to Ubuntu 16, there were no issues encountered during the installation of Wizard 2.0 on Ubuntu 22.04. Additionally, the tool was successfully tested and found to work with Docker on Ubuntu 22.04. This demonstrates the tool’s compatibility with newer operating systems despite the outdated official support documentation. The following are steps for installation and setup:

- Download Dynamixel Wizard file for Ubuntu: **emanual.robotis.com**
- Then open terminal and type the following commands:
  ```
  $ sudo chmod 775 DynamixelWizard2Setup_x64
  $ ./DynamixelWizard2Setup_x64
  $ sudo usermod -aG dialout <your_account_id>
  $ reboot
  ```
  Note: your_account_id is the name of your PC, usually written in terminal.
- Open Dynamixel Wizard as seen on Fig. 1.
- Plug in U2D2 communication interface with three pin cable (TTL). Connect USB to PC and Power Supply 12V with 1A.
- Click on ”options” and select the scanning options for Dynamixel motors as seen on the Fig. 2.
- There should be one motor showing up as in Fig. 3.
- Try rotating the motor with Goal Position, you will see the Joint position change and can see the minimum and maximum angles for this motor. Example: 287 - 78.84 angles.
- Connect all other motors in series and do search. You should see them all after searching as in Fig. 4.

B. Install dynamixel software development kit

The DYNAMIXEL SDK is a software development kit that provides control functions for DYNAMIXEL actuators and platforms via packet communication. The SDK’s API is



DYNAMIXEL Wizard 2.0 - v2.1.7.7

ROBOTIS | DYNAMIXEL | e-Manual | Forum

Tutorials

- Introduce Minions Clock (with OpenRB-150)
- Writing Minion Dave (feat. OpenRB-150, XL330-M288)
- OpenRB-150 : Arduino Compatible DYNAMIXEL controller
- DYNAMIXEL Quick Start Guide with Arduino
- DYNAMIXEL Quick Start Guide with DYNAMIXEL Wizard 2.0

News

1. [Product of the Month] XW430-T333-R

DYNAMIXEL XW series is IP68 rated (1m, 24hr) waterproof, making it ideal for use in humid environments or applications requiring water resistance.

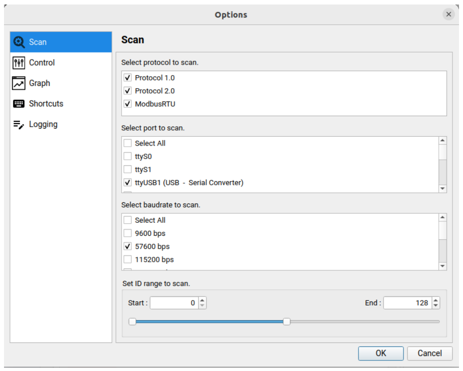
● Link to e-shop : (Link)

2. [Video of the Month] Ringbot

Introducing Ringbot developed by KIMLAB. This robot has been inspired by Star Wars, Ringbot was recently featured in the IEEE Transactions on Robotics. Learn more about Ringbot at the following link below.

● Link to Video : (Link)

Figure 1. Dynamixel Wizard 2.0



Options

Scan

Select protocol to scan.
- [✓] Protocol 1.0
- [✓] Protocol 2.0
- [✓] ModbusRTU

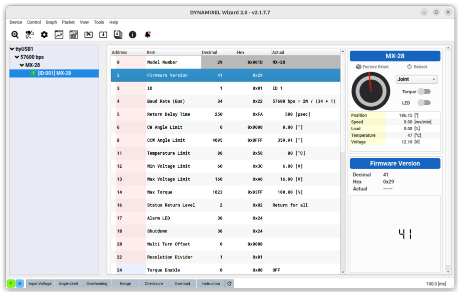
Select port to scan.
- [ ] Select All
- [ ] ttyS0
- [ ] ttyS1
- [✓] ttyUSB1 (USB - Serial Converter)

Select baudrate to scan.
- [ ] Select All
- [ ] 9600 bps
- [✓] 57600 bps
- [ ] 115200 bps

Set ID range to scan.

Start : 0 End : 128

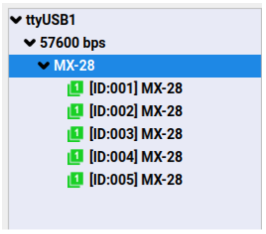
Figure 2. Options for searching Dynamixel Motors. Protocol 1.0, ttyUSB*, and 57600 baudrate.



DYNAMIXEL Wizard 2.0 - v2.1.7.7

Address	Item	Decimal	Hex	Actual
0	Model Number	29	0x001D	MX-28
2	Firmware Version	41	0x29	
3	ID	1	0x01	ID 1
4	Baud Rate (Bus)	34	0x22	57600 bps = 2M / (34 + 1)
5	Return Delay Time	250	0xFA	500 [μsec]
6	CW Angle Limit	0	0x0000	0.00 [°]
8	CCW Angle Limit	4095	0x0FFF	359.91 [°]
11	Temperature Limit	80	0x50	80 [°C]
12	Min Voltage Limit	60	0x3C	6.00 [V]
13	Max Voltage Limit	160	0xA0	16.00 [V]
14	Max Torque	1023	0x03FF	100.00 [%]
16	Status Return Level	2	0x02	Return for all
17	Alarm LED	36	0x24	
18	Shutdown	36	0x24	
20	Multi Turn Offset	0	0x0000	
22	Resolution Divider	1	0x01	
24	Torque Enable	0	0x00	OFF

Figure 3. The Dynamixel motor showing up in Dynamixel Wizard. The configurations can be set up here. When the motor overheat, repair it with "Firmware Management" by completing the instructions on screen. (Model: MX Series, MX 28, V41, Port: ttyUSB*)



ttyUSB1
- 57600 bps
  - MX-28
    - [ID:001] MX-28
    - [ID:002] MX-28
    - [ID:003] MX-28
    - [ID:004] MX-28
    - [ID:005] MX-28

Figure 4. All five motors connected and found by Dynamixel Wizard.

specifically designed for DYNAMIXEL hardware, making it an essential tool for DYNAMIXEL-based projects.

Prerequisites: To effectively utilize the SDK, a solid understanding of the C/C++ programming language is required.

Installation and Workspace Setup: The installation process involves creating a workspace and downloading a Git repository with DYNAMIXEL software examples. The necessary commands for these steps are as follows:

```
$ sudo apt-get install ros-humble-dynamixel-sdk
$ mkdir -p ~/robotis_ws/src
$ cd robotis_ws
$ git clone --depth 1 -b humble-devel https://
github.com/ROBOTIS-GIT/DynamixelSDK.git
$ colcon build --symlink-install
$ source /opt/ros/humble/setup.bash
$ . install/local_setup.bash
```

SDK Usage Steps:

- **Connect the USB Port:** Insert the USB cable and ensure the appropriate port is detected. Typically, the device is available as ttyUSB0.
- **Verify and Grant Port Permissions:** Use terminal commands to identify the connected port and set the required permissions for communication.
- **Run Example Code:** Execute the example code included in the SDK, which:
  1.Handles data requests on the /get_position topic;
  2.Creates the /set_position topic to control each motor.
- **Adjustments for Snake Robot Configuration:** The default example code was modified to accommodate the specific configurations of the snake robot. Without these adjustments, running the code could result in overheating errors, requiring a complete reinstallation and recalibration of each motor.

To use the SDK, type the following commands into the terminal:

```
$ ls /dev/tty*h
$ sudo usermod -a -G dialout <your_account_id>
$ ros2 run dynamixel_sdk_examples
read_write_node
in another terminal:
$ ros2 topic pub -1 /set_position
dynamixel_sdk_custom_interfaces/msg/
SetPosition "{id: 5, position: 3073}"
```

By following these steps and making the necessary configuration changes, the DYNAMIXEL SDK can be effectively utilized for controlling the snake robot.

Observe the change in position of the motor with chosen id and position. Example is shown in Fig. 5.

III. Moving snake robot joints with code

A. Mapping formula

In order to make meaningful commands to the motor of the snake robot, the mapping formula was created to convert the position numbers to angles reachable for robot joints:

$$\text{position} = 2048 - \text{angle\_degrees} \times 11.3888 \qquad (1)$$

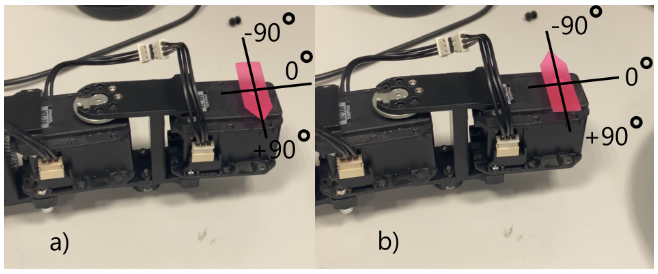


Figure 5. SDK example movement with terminal: a) before; b) after the command execution. The marker was additionally applied to distinguish motor 5 position. Motor moved from -90 deg (position 1023) to +90 deg (position 3073).

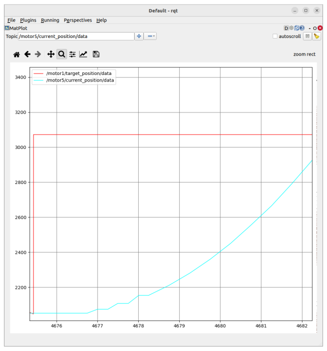


Figure 6. Step response of the joint 5 for movement from +90 to 0 degrees.

Position is the message type that is handled by read_write_node and can be parsed to set_position topic to move the snake robot's joints. The angle_degrees is the angle of the joint in degrees for the joints. The Fig. 5 represents the degrees convention used for the formula (1).

A node for moving all joints with the code was created. It is located in the package with the name 'move.py' along with other source codes. The 'move.py': Python script which controls the movements of a snake robot's joints using ROS2 and Dynamixel motors.

ROS 2 Node Initialization: The class `JointController` inherits from the ROS2 `Node` class and is initialized as a ROS node named `'joint_controller'`. A publisher is created to send position commands to the topic `/set_position`, which controls the positions of the robot's joints. A service client is created to request the current position of a joint using the `/get_position` service.

Angle to Position Conversion: The function `angle_to_position` converts an angle in degrees into a Dynamixel-compatible position value. This is necessary because Dynamixel motors use position values ranging from 0 to 4095, with 2048 being the center (neutral) position. The Formula (1) is used. The reverse conversion, from position to angle, is done by `position_to_angle`, which converts a motor position back to degrees.

Setting Joint Positions: The `set_joint_positions` method publishes position values for multiple joints (up to 5 joints). It takes a list of joint angles (in degrees), converts each angle to a position value using the `angle_to_position` function, and sends the position to the corresponding joint via the `/set_position` topic. After each joint's position is set, the method logs the action and waits for 200 milliseconds before moving to the next joint, allowing for smoother movements.

Enabling Torque: The `enable_torque` method publishes commands to enable the torque on specified joints. Enabling torque activates the motors, allowing them to hold their position or move as commanded.

Main Function:

- In the `main` function, ROS2 is initialized, and an instance of `JointController` is created.
- It enables torque for five joints (`joint_ids = [1, 2, 3, 4, 5]`).
- It sets the angles of these joints to `0` degrees using the `set_joint_positions` method.
- The program handles a keyboard interrupt to safely shut down and destroy the ROS node.

This code could have a tutorial "insert a code here" for students to complete 'move.py' and learn basics of topics, nodes with dynamic environment. Later, we used this script as a move_to_start command as in regular manipulators' packages, since all joints can be brought to 0 degrees with this script.

IV. Step response

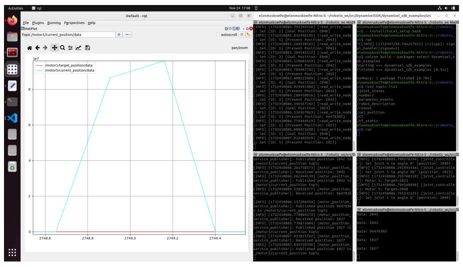
Essential skill in robotics is to use graphs and plots to understand the behavior of the data coming from a robot. In this project, we used rqt plotting in the same manner as in ROS1. The topics are written directly to the rqt GUI. Using rqt, we tracked the target position and real joint position for a step response for Lab #3 of the course.

In order to implement the behavior, new nodes were created: "simple_pub.py" and "step_response.py". The first one is a node that subscribes to `get_joint_position` and publishes the position data to a new topic `/motor#/current_position`. This was done since the rqt program cannot handle the message type from `get_joint_position` directly, and the new topic has message type of INT32. The target position of a joint was upldated in "simple_pub.py". The step response of the joint for moving 90 degrees can be seen on Figure 6.

A challenge we faced in this step was that some values from the `get_joint_position` topic were too high for real controller to handle, therefore, we low pass filter for positions. This behavior might be due to the hardware connection failure for high speed movements and data loss. The representation of data loss can be seen from Figure 7.

V. Moving joints in a wave manner

In Laboratory #3, students were tasked with programming the robot to move in a wave-like pattern. This movement was


Activities
rqt
Nov 24 17:08
Default - rqt
File Plugins Running Perspectives Help
MatPlot
Topic /motor5/current_position/data
autoscroll
pan/zoom
1e7
/motor1/target_position/data
/motor5/current_position/data
0
2
4
6
8
2748.6
2748.8
2749.0
2749.2
2749.4

```
elonmuskswife@elonmuskswife-Nitro-5: ~/robotis_ws/src/DynamixelSDK/dynamixel_sdk_examples/src
```

```
[INFO] [1732450084.792592510] [read_write_node]: Set [ID: 2] [Goal Position: 2048]
[INFO] [1732450085.032446195] [read_write_node]: Get [ID: 5] [Present Position: 2042]
[INFO] [1732450085.288538543] [read_write_node]: Get [ID: 5] [Present Position: 2042]
[INFO] [1732450085.304545522] [read_write_node]: Set [ID: 3] [Goal Position: 2048]
[INFO] [1732450085.528601454] [read_write_node]: Get [ID: 5] [Present Position: 2042]
[INFO] [1732450085.784407792] [read_write_node]: Get [ID: 5] [Present Position: 2042]
[INFO] [1732450085.800522889] [read_write_node]: Set [ID: 4] [Goal Position: 2048]
[INFO] [1732450086.024491408] [read_write_node]: Get [ID: 5] [Present Position: 2042]
[INFO] [1732450086.280458016] [read_write_node]: Get [ID: 5] [Present Position: 2042]
[INFO] [1732450086.296567242] [read_write_node]: Set [ID: 5] [Goal Position: 1023]
[INFO] [1732450086.536367063] [read_write_node]: Get [ID: 5] [Present Position: 96470305]
[INFO] [1732450086.776444519] [read_write_node]: Get [ID: 5] [Present Position: 1027]
[INFO] [1732450086.808421690] [read_write_node]: Set [ID: 1] [Goal Position: 2048]
[INFO] [1732450087.032402919] [read_write_node]: Get [ID: 5] [Present Position: 1027]
```

```
elonmuskswife@elonmuskswife-Nitro-5:~/robotis_ws$ . install/local_setup.bash
elonmuskswife@elonmuskswife-Nitro-5:~/robotis_ws$ rqt
^C[INFO] [1732447250.746257923] [rclcpp]: signal_handler(signum=2)
elonmuskswife@elonmuskswife-Nitro-5:~/robotis_ws$ colcon build --packages-select dynamixel_sdk_examples
Starting >>> dynamixel_sdk_examples
Finished <<< dynamixel_sdk_examples [0.51s]

Summary: 1 package finished [0.70s]
elonmuskswife@elonmuskswife-Nitro-5:~/robotis_ws$ ros2 topic list
/joint_states
/numbers
/parameter_events
/robot_description
/rosout
/set_position
/tf
/tf_static
elonmuskswife@elonmuskswife-Nitro-5:~/robotis_ws$ rqt
```

```
service_publisher]: Published position 2042 to /motor5/current_position topic
[INFO] [1732450086.281758573] [motor_position_service_publisher]: Received position: 2042
[INFO] [1732450086.282494539] [motor_position_service_publisher]: Published position 2042 to /motor5/current_position topic
[INFO] [1732450086.536928377] [motor_position_service_publisher]: Received position: 96470305
[INFO] [1732450086.537206454] [motor_position_service_publisher]: Published position 96470305 to /motor5/current_position topic
[INFO] [1732450086.778004574] [motor_position_service_publisher]: Received position: 1027
[INFO] [1732450086.778871004] [motor_position_service_publisher]: Published position 1027 to /motor5/current_position topic
[INFO] [1732450087.033825792] [motor_position_service_publisher]: Received position: 1027
[INFO] [1732450087.034759396] [motor_position_service_publisher]: Published position 1027 to /motor5/current_position topic
```

```
[INFO] [1732450085.790278344] [joint_controller]: Set joint 4 to angle 0° (position: 2048)
[INFO] [1732450086.291991436] [joint_controller]: Set joint 5 to angle 90° (position: 1023)
[INFO] [1732450086.793768318] [joint_controller]: Motor 5: Target=1023
[INFO] [1732450086.794560998] [joint_controller]: Motor 5: Target=2048
[INFO] [1732450086.795332335] [joint_controller]: Set joint 1 to angle 0° (position: 2048)
```

```
data: 2042
---
data: 2042
---
data: 96470305
---
data: 1027
---
data: 1027
---
```

Figure 7. Data loss visible in rqt plot.

successfully replicated on a real robot by assigning specific angles to each joint. Depending on their position in the robot's structure, the joints were programmed to oscillate between -30 and 30 degrees, or 30 and -30 degrees. The motion was looped continuously to produce a repetitive wave-like movement. The code is included inside the main function of 'step_response.py' file.

While the snake robot executed the wave motion, an rqt plot was used to monitor and display the target and current positions of its joints. However, a recurring issue with data loss was observed in the plot. As shown in Figure 8, the target position of Joint 5, represented by the red line, oscillates smoothly between -30 and 30 degrees with short delays introduced by sleep commands in the code. In contrast, the blue line, representing the actual current position, remains static for specific time intervals even though the robot continues to move.

This mismatch highlights a data loss problem, likely caused by communication issues within the ROS2 framework. Potential causes for this issue include limited bandwidth, buffer overflows, or synchronization challenges between the robot's sensors and the control system.

To address this, it is crucial to investigate the ROS2 communication stack, specifically the transport layer and node configurations. Optimizations such as adjusting the message publishing rate, increasing the queue size, or using alternative middleware implementations could help mitigate this data loss. Ensuring that the robot's controllers and sensors are properly synchronized and that hardware resources are not overburdened is also essential for reliable real-time operation.

VI. URDF and Creation of the Package

The URDF (XACRO) files originally created for Noetic ROS1 were not directly compatible with the ROS2 framework. As a result, there was a need to generate updated URDF files from the robot_description .STL files, which are provided on the repository fenixkz/ros_snake_robot.

Initial attempts to adapt and visualize the robot model in rviz2 were unsuccessful. These failures highlight inconsistencies or mismatches within the XACRO files, which must be thoroughly reviewed and modified to ensure proper compatibility with ROS2. Resolving these issues will allow the robot model to function correctly within the ROS2 ecosystem, including visualization in rviz2.

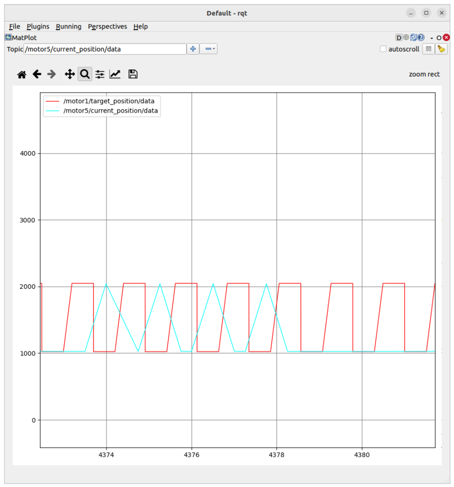


Figure 8. Moving the snake robot in a wave manner.

To streamline this process, all the related code and files have been organized into their respective folders. A comprehensive package was created, incorporating all the necessary files for students to use in their laboratory assignments.

The complete codebase and package for this project are available on GitHub and can be accessed via the following link: GitHub.

VII. Conclusion

A systematic approach was employed to ensure a successful transition from ROS1 to ROS2, while aligning with the project's technical and educational objectives. This structured process not only addressed compatibility issues but also laid the groundwork for future enhancements and applications.

Future steps could include incorporating package installation for the simulation version of the robot and enabling it to execute the same movement commands within rviz2. Another key milestone would be the integration of the MoveIt library to facilitate Direct and Inverse Kinematics for the snake robot in ROS2, enabling more advanced motion planning and control.

These enhancements will further improve the robot's functionality, making it a more versatile tool for both learning and research purposes in robotics.