Goal:
In the previous tutorial, you learned about the basics of ROS 2. Based on that knowledge, the goal of this tutorial is to learn how to work with the AiNex robot. You will send simple motion commands (position control) to the robot and process the received camera images.

Robot Setup and Basic Examples:

1) Make sure you know enough about the following ROS 2 basics: nodes, topics, messages, services, and basic commands (*e.g.* colcon *build*, *ros2 run*, *ros2 topic*, ros2 node *etc.*). Check the ROS online tutorial (https://docs.ros.org/en/jazzy/Tutorials.html) for further information.

2) Turn on the robot. There are a few ROS nodes that are launched automatically on boot. Once the robot is on, check if you have connection with the robot with
   $ ping 192.168.50.20x
   (Note: The last digit of the IP address is the ID of your robot (1-5))
   Once you have the network connection, you can check the running nodes with "*ros2 node list*" or with "*rqt_graph*" in your terminal. You should be able to see two nodes named "/Joint_Control" and "/camera_publisher". Sometimes the robot does not start properly, if you can ping the robot but cannot see the nodes, try to reboot the robot.

3) Download the template workspace from Moodle, build and source the workspace
   $ colcon build –symlink-install
   $ source install/setup.bash
   The workspace contains three packages: *ainex_motion*, *ainex_vision* and *servo_services*.
   - A *joint_controller.py* node located in the *ainex_motion* package contains a few functions that sends commands (through topics) to, and request information (through services) from the servo controller on the robot side. There are a few examples in the main function to show you how to use these functions. Run the node to see how the example work:
     $ ros2 run ainex_motion joint_controller
     Feel free to comment/uncomment different parts of the main function to try out different demos.
   - The customized communication interfaces (msgs and srvs) are defined in the package servo_services.
   - The ainex_vision package contains a node that subscribe camera frames that the robot publishes. Run the node to visualize the camera stream:
     $ ros2 run ainex_vision camera_sub

   Once you are running the nodes, you can run *rqt_graph* again to check the communication between them.

4) Apart from the function calls in the joint_controller.py. You can also call the services /Lock_All_Joints and /Unlock_all_joints directly with terminal commands
   $ ros2 service call /Lock_All_Joints std_srvs/Empty {}
   $ ros2 service call /Unock_All_Joints std_srvs/Empty {}
   Check if you can move the joints manually after the service call.

5) Now the robot and basic examples are running. Let's do some exercise.

Exercises:

Exercise 1: Joint Position Visualization

1) Open the file joint_visualization_node.py located in the *ainex_motion* package, fill in the gaps marked with TODOs. This node should repetitively call the *getJointPositions()* provided in the joint_controller module to retrieve the real-time joint positions and publish them as the standard sensor_msgs/JointState message. Don't forget to add the entry point for this node in the *setup.py* file

2) Visualize the message with plotting tools like Plotjuggler.
   You can install Plotjuggler's ROS package with
   $ sudo apt update && sudo apt install ros-$ROS_DISTRO-plotjuggler-ros
   Run with
   $ ros2 run plotjuggler plotjuggler
   Click the Start button under the Streaming panel and choose the topic you want to visualize.

3) While plotjuggler is plotting the joint positions, move the arms a little, and observe the effects. Make a screenshot of the plots of the 2 DOF shoulder and the 2 DOF elbow joints.
   Hint: In later exercises, you can also use this node to identify joint names, and check appropriate joint motion range before you command motions on robot.

Exercise 2: Motion Command

1) Implement two nodes that perform the following task (Note: you could either directly add new nodes to the *ainex_motion* package, or create a new package for the dedicated functionality. Don't forget to add the *ainex_motion as* the dependencies of your new package if you do the latter) :
   - *keyboard_input_node.py*: Takes in number input 1-4 on the keyboard and publish it as a std_msgs/Int16 message. Note that it should only publish the integer numbers 1-4 and ignore all other input on the keyboard.
   - *ainex_motion_node.py*: Subscribe to the keyboard command, and instantiate a JointController to communicate with the robot. (Hint: check how it is done in *joint_visualization_node.py*).
     The robot should start with a *crouch* posture. Upon receiving command 1-4, activate different motion sequence on the robot:
     - Command 1: The AiNex robot should move its left arm to a home position. Feel free to define any SAFE home position for your robot.
     - Command 2: The AiNex robot should do a repetitive arm motion with its left arm, starting from home position. Feel free to create any kind of repetitive arm motion. e.g. waving motion.
     - Command 3: The AiNex robot should move its right arm in addition to its left arm, simultaneously mirroring the motion of its left arm.
     - Command 4: The AiNex robot should unlock all joints.

2) Run the nodes and visualize the node communication with *rqt_graph*.

3) Make a demo video of your robot performing the task above.

Exercise 3: Image processing with OpenCV

1) The example code of *camera_sub.py* demonstrates how to subscribe to the CompressedImage message published by the robot, and decompress with the *cv2.imdecode()* function. Read the code and understand the general structure. Also familiarize yourself with basic image processing with OpenCV. There are tons of tutorials online.
2) Expand the *camera_sub.py* node to implement the following task: Process the decompressed frame to extract purple color blobs. For this purpose, transform the image into HSV colour space. Take the largest color blob and calculate the position of its center in pixel coordinates. A purple ball is provided to each group to test the performance of the color blobs detection algorithm Output this blob position on the terminal or via plotjuggler.

3) Make a demo video to show your result.

Results to submit:

1) The **/src** folder of your workspace, containing all your source code.
   Note: **Comment** your code. Also, write **your names** at the beginning of the source code file.

2) Screenshot of Exercise 1.3.

3) Demo videos showing the result of Exercise 2 and 3.

Compress all the required results into a .zip or .tar.gz file.
Naming convention:

LastNameOfStudent1_LastNameOfStudent2_LastNameOfStudent3_T*number*.tar.gz

Try to keep the video size compact (below 30MB). You can use Handbrake (https://handbrake.fr/) or ffmpeg to reduce the video filesize. Please keep the length to a maximum of 60 seconds, a minimum framerate of 25 Hz and a minimum resolution of 720p (1280x720).

Submit the file containing your results **before** the deadline to the BILHR **MOODLE**

Evaluation of your work:
Your work will be evaluated based on your results (required code and other documents).

Deadline: ***21*.05.2024, 23:59**