Robot Practical Course Bachelor

Assignment #1

In this assignment you will setup the software framework ROS which will be used throughout this course. You will get to know the fundamentals of the framework using the tutorial-simulator "turtlesim". The goal is to draw a "Haus vom Nikolaus" (an old german drawing puzzle for kids) in the simulator.

NOTE: LaTeX renders the tilde (~) as a different character. Do not copy it into your shell.

Task 1.1 Set up ROS 2 container environment: To be more independent of the host system, we will use a container environment to run ROS 2 on your PC. This way, you can run the same code on different systems without worrying about different linux distributions or what software is installed on the host system. This task will guide you through the setup of the development environment for the practical course.

1.1.1: Boot the lab computer to Linux and login using your normal account (e.g. `6musterm`).

1.1.2: Open a terminal and type the following command to clone the repository containing our ROS 2 container environment:

```
git clone git@github.com:TAMS-Group/rpc_workspace.git
```

1.1.3: Open VSCode and open the the folder/repository you just cloned.

1.1.4: Go to the extensions tab (the square icon on the left sidebar), search for the extension "Dev Containers" and click on the install button.

1.1.5: Press F1 (or Ctrl+Shift+P) to open the command palette and type "Dev Containers: Reopen in Container".

1.1.6: Wait for the container to build and start. This might take a while, so be patient.

1.1.7: Run the following command in a terminal on the host system to enable GUI forwarding from the container to the host system:

```
    echo 'xhost local:root' >> ~/.bashrc
```

1.1.8: Now you are ready to go. To verify that everything is working, run the following command in a terminal inside the container:

```
    rviz2
```

Task 1.2 Start turtlesim: In this section you will learn how to start ROS 2 nodes. You will teleoperate a turtle in turtlesim and get to know basic ROS 2 commands.

1.2.1: Open a terminal in the container environment and type the following command to start the turtlesim node:

```
ros2 run turtlesim turtlesim_node
```

This brings up the turtlesim environment with one turtle in it. To teleoperate this turtle you can start another node in a new terminal to send commands for the turtle:

```
ros2 run turtlesim turtle_teleop_key
```

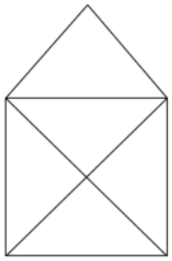
turtlesim should be started and you should be able to move the turtle around by using the arrow keys.
Now try some of these commands in the 4th terminal to learn about the environment you just started:

```
ros2 node list
ros2 topic list
ros2 topic info <TOPICNAME>
ros2 topic echo <TOPICNAME>
ros2 service list
ros2 run rqt_graph rqt_graph
```

Task 1.3 Explain ROS: Explain first to each other and then to a supervisor the basic concepts of ROS. You should name:

- Nodes
- Communication
  - Messages
  - Topics
  - Services

Task 1.4 Create your first ROS node: Write a node that moves the turtle around, such that it draws a ”Haus vom Nikolaus”. It looks like this and has to be drawn in one move.



1.4.1: Create a ROS 2 package with your group's name in your workspace.

What is a package? A package is a directory that contains all the files needed to build one or more ROS nodes and to run them. It contains the source code, the build instructions, and requirements (e.g. dependencies on other packages).

There are two main programming languages used in ROS 2: C++ and Python. We recommend you choose Python for this task, as it is easier to get started with and has a more straightforward syntax for beginners.

To create a Python package you can use the following command:

```
cd src && ros2 pkg create --build-type ament_python --node-name my_node <YOUR_GROUP_NAME>
```

Make sure to replace <YOUR_GROUP_NAME>with your actual group name.

This will create a folder in the ‘src‘ directory of your workspace with the name of your group containing the following files:

- package.xml: This file contains metadata about your package, such as its name, version, and dependencies.
- setup.py: This file is used to build and install your package. It specifies the entry point for your node (the function that will be executed when the node is started) and what additional files should be installed during the installation process.
- setup.cfg: This file is needed if we want to run our node using the ‘ros2 run‘ command.
- A folder with the same name as your group: This folder will contain your Python code for the node. In this case it contains a file called ‘my_node.py‘ with the entrypoint function already defined.

1.4.2: Customize the package.xml file to include the dependencies you need for your node.

Dependencies can be changed later on but it is easier to specify them from the start. Your program will depend on the ‘turtlesim‘ and the ‘geometry_msgs‘ packages.

Add them to the ’package.xml’ file like this:

```
        <depend>turtlesim</depend>
        <depend>geometry_msgs</depend>
```

This way the everybody knows what needs to be installed or compiled before running your node.

ROS 2 uses ‘colcon‘ as the main build-system for packages. Sadly the ‘colcon‘ command line interface is quite verbose, and you need to be careful to run it in the right directory. Otherwise you will get tricky errors that are hard to debug. We therefore have added a ‘justfile‘ (similar to a Makefile) to the workspace that contains the most common commands you will need to build and deploy your packages.

To build your package, run the following command in the root of your workspace:

```
just build
```

You can also build a specific package by running:

```
just build-package <YOUR_GROUP_NAME>
```

1.4.3: Look up the ROS tutorial for a simple publisher

`https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html#write-the-publisher-node`

and use it as a template for your node.

Do **not**:

- Follow it exactly, as file names etc. can be different
- Create a second package, use the one you created in the previous task.
- Run colcon build in the src directory, it will not work. Instead use the ‘just build‘ command in the root of your workspace.

Use it as inspiration to write your own node that publishes messages to control the turtle in turtlesim later on.

1.4.4: Now write a node that publishes messages that make the turtle draw a ”Haus vom Nikolaus”. In this tutorial you can look up which topic the turtle uses:

`https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html`

Every package for ROS has to be build at least once in order to make it known to the workspace. This has to be done even if you implement the node within the package in python. Afterwards you need to open a new terminal and start your node with this command (just as you started the nodes in the previous task):

```
ros2 run <ROS_PACKAGE> <THE_PROGRAMM>
```