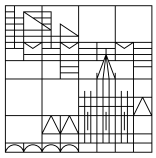
Task Sheet 1

Universität Konstanz



Introduction to ROS 2

Deadline 10:00am April 19, 2024

Review on April 23 & 24, 2024

Lecture: *Advanced Autonomous Robotics*, Summer Term 2024

Lecturer: Lecturer: Prof. Dr.-Ing. Heiko Hamann

Tutor: Jonas Kuckling & Paolo Leopardi

The Robot Operating System (ROS)[1] is a set of software libraries and tools that simplify the development of applications for robots. It does this through hardware abstraction, device drivers, inter-process communication and package management, among other things. ROS is not only used in research and development, but is also in industry. ROS 2 is the further development of ROS and integrates previously missing requirements, such as real-time capability and security.

In this tutorial, you will learn the basics of ROS 2 using Python 3.

In this tutorial, you will set up your ROS 2 development environment and program your first ROS 2 nodes. As the TurtleBot 4 runs on ROS 2 Humble only, this exercise sheet guides you through the installation of this ROS 2 version. ROS 2 Humble and the required TurtleBot 4 packages are only available for Ubuntu 22.04 LTS (Jammy Jellyfish).[2] You can either install everything on your own PC, in a virtual machine, or use docker. However, we highly recommend a native installation (not in a virtual machine or docker), as this facilitates the networking set up. If you choose to use a guest system (VM, Docker, ...) make sure to configure the network in *bridged* mode.

Task 1.1 Installation of ROS 2 (prepare at home)

Install ROS 2 Humble following these instructions:
`https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html`.
Install the packages `ros-humble-desktop` and `ros-dev-tools`.

Then run the talker-listener example on the same page and submit a screen shot showing the output of the talker and listener on ILIAS.

Task 1.2 ROS 2 Tutorials (prepare at home)

Do the following tutorials in the "Beginner" section on the website `https://docs.ros.org/en/humble/Tutorials.html`:

- Configuring your ROS 2 environment (but do not set the `ROS_LOCAL_HOST` environmental variable, as it will interfere with connecting to the TurtleBots in future exercises)
- Using turtlesim and rqt
- Understanding nodes
- Understanding topics
- Understanding services

[1] `https://docs.ros.org/en/humble/index.html`

[2] ROS 2 can also be installed from source on a variety of systems. However, we cannot offer any troubleshooting assistance if you do not install the pre-built packages under Ubuntu 22.04 LTS.

- Understanding parameters

These tutorials will introduce concepts that will be relevant during the following practical exercises. You will have the opportunity to ask any questions you might have about these concepts in the practical tutorial session.

Task 1.3 Your first ROS 2 package (practical tutorial)

In this exercise, you will follow the tutorials guiding you to implement your first ROS 2 package. Do the following tutorials in the "Beginner" section on the website `https://docs.ros.org/en/humble/Tutorials.html`:

- Creating a package
- Writing a simple publisher and subscriber (Python)
- Writing a simple service and client (Python)

Show your results of this exercise to your tutor to pass the practical tutorial session.

Task 1.4 An advanced ROS 2 package (practical tutorial; optional)

This exercise is optional and is not required to pass the practical tutorial session. Design a more complex ROS 2 package to play the towers of Hanoi. The package should do the following:

- Create a node (HanoiSensor) that pretends to be a sensor. It checks the keyboard for inputs and publishes relevant inputs on a topic called `hanoi_input`.
- Create a second node (HanoiController) that listens to the topic `/hanoi_input`. If it receives any input of 1,2,3 this selects the chosen stick. If it receives the first number, the node should treat it as the stick of origin. Once it receives the second number, it should publish a message to the topic `/hanoi_move`. If the `/hanoi_input` is the letter "C" it should clear the currently stored numbers and treat the next received number as if it was the first received number. If it receives the letter "R", it should call a service on the HanoiWorld node (see next bullet) to reset the world.
- Create a third node (HanoiWorld) that listens to the topic `/hanoi_move`. If it receives a move, it should perform the corresponding move and print the current state of the towers of Hanoi to the terminal. The node should also offer a service that allows resetting the world to its initial state.