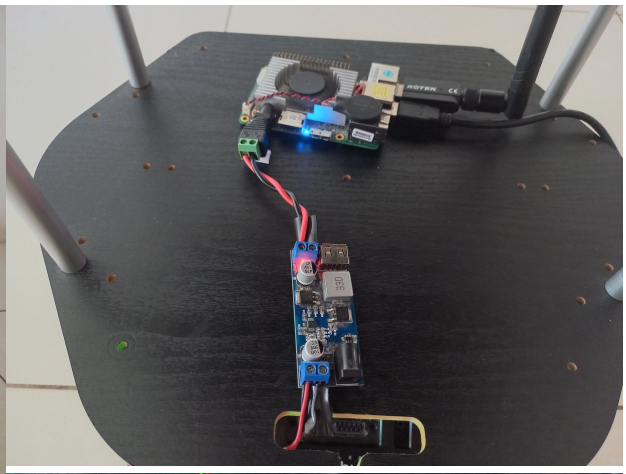
Robotics. And other stuff too.

FEBRUARY 20, 2024FEBRUARY 20, 2024 OPEN-SOURCE, ROBOTICS LEAVE A COMMENT
ROS ROS 2 TURTLEBOT

ROS 2 on Kobuki TurtleBot

TurtleBots are relatively low cost robot platforms intended to run the open source Robot Operating System (ROS) software. There have been several series of TurtleBots (https://www.turtlebot.com/), the latest being TurtleBot 4 which is built upon the iRobot Create3 mobile base.

The TurtleBot 2 is an older generation platform built upon the Kobuki mobile base. The Kobuki platfor (https://github.com/yujinrobot/kobuki) is well supported in ROS Kinetic with deb packages available fo the drivers and most common sensors. Upgrading TurtleBot 2 to ROS 2 Humble – in February 2024 – presented some challenges which are discussed in this article.



The setup outlined in this article can be downloaded from my turtlebot2_ros2 Github repository (https://github.com/idorobotics/turtlebot2_ros2).

Kobuki driver

Upgrading from ROS 1 Kinetic to ROS 2 Humble requires an OS upgrade from Ubuntu 16.04 to Ubuntu 22.04, as well as learning ROS 2 concepts.

In our setup, the code is running on the AAEON UP-CHT01 Board (https://www.aaeon.com/en/p/up-board-computer-board-for-professional-makers) which has only 4GB memory, 1.44GHz x 4 Intel Atom processors with Intel HD Graphics, and 32GB disk capacity. Due to these limitations, only install the *ros-base* and *dev-tools* packages instead of the full *ros-desktop* installation. Additional packages can be compiled or installed as required.

For the Kobuki base, the only packages available via Ubuntu package manager were the ROS interfaces (https://github.com/kobuki-base/kobuki_ros_interfaces) and velocity smoother (https://github.com/kobuki-base/kobuki_velocity_smoother). These can be installed from terminal using `sudo apt-get install ros-humble-kobuki-ros-interfaces ros-humble-kobuki-velocity-smoother`.

In a ROS 2 workspace (overlay), clone the following packages required for compiling the Kobuki driver:

- kobuki_core (https://github.com/kobuki-base/kobuki_core)
- kobuki_ros (https://github.com/kobuki-base/kobuki_ros)
- kobuki_ros_interfaces (https://github.com/kobuki-base/kobuki_ros_interfaces) (if not installed via apt)
- cmd_vel_mux (https://github.com/kobuki-base/cmd_vel_mux)
- ecl_core (https://github.com/stonier/ecl_core)
- ecl_lite (https://github.com/stonier/ecl_lite)

The sophus library (https://github.com/stonier/sophus) package can be installed by running `sudo apt-get install ros-humble-sophus`. Building the workspace with this dependency however shows a list of errors of type ***ISO C++11 requires at least one argument for the "..." in a variadic macro"*** . Fortunately, this is a known issue (https://github.com/kobuki-base/kobuki_core/issues/49) but the solution PR was not merged (https://github.com/stonier/sophus/pull/23) at the time of writing. A temporary solution is to manually make the changes described in the commit files (https://github.com/stonier/sophus/pull/23/files) since compiling the sophus package from source proved to be too memory intensive on the UP Board.

Install any dependencies using `rosdep install -i --from-path src --rosdistro humble -y` and remember to either build packages individually using the `--packages-select [package_name]` flag or sequentially using the `--executor sequential` flag when running `colcon build --symlink-install`.

Follow the official guide in the kobuki documentation (https://kobuki.readthedocs.io/en/release-1.0.x/software.html) to update *udev rule* for the USB connection, check the version information, and run the *kobuki-simple-keyop* test node to ensure that the robot can move.

ROS 2 teleoperation

Teleoperation allows the robot to be controlled remotely, from a distance. The ROS 2 teleoperation nod publish to */cmd_vel* but Kobuki expects input from the */commands/velocity* topic so a remapping is required. The goal is to control the robot from a laptop (workstation) that is not physically connected to the robot.

Communication setup

Install the following apt packages on the workstation and robot:

- ros-humble-teleop-twist-keyboard (https://index.ros.org/p/teleop_twist_keyboard/github-ros2-teleop_twist_keyboard/)
- ros-humble-joy-teleop (https://index.ros.org/p/joy/)
- ros-humble-teleop-twist-joy (https://index.ros.org/p/teleop_twist_joy/github-ros2-teleop_twist_joy/)

ROS 2 uses a Discovery Server protocol which replaces rosmaster (http://wiki.ros.org/rosmaster) from ROS 1. This discussion thread (https://discourse.ros.org/t/new-discovery-server/17383) explains the ide behind this. ROS 2 nodes on the same domain automatically discover each other, enabling two-way communication. Ensure that *ROS_DOMAIN_ID* is set to the same value on both the robot and workstation, to enable DDS communication.

Start robot

Open an ssh connection to the robot and start the robot by running `ros2 launch kobuki_node kobuki_node-launch.py` and the startup tune will play.

The robot is now ready to receive control commands.

Keyboard teleoperation

Keyboard teleoperation sends command signals to the robot using the keyboard. This is especially usef when debugging.

Open a terminal on the workstation and run `ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args --remap cmd_vel:=commands/velocity`.

The robot can now navigate using the keyboard.

Joystick teleoperation

The goal of joystick teleoperation is to control the robot from a separate workstation – at a distance – using the PDP joystick pictured below.

The ROS 2 joy package interfaces a generic joystick to ROS 2. Follow the instructions to configure a joystick (http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick) and test the setup by running the `ros2 run joy joy_node` command and echoing the *joy* topic to observe the output when pressing the joystick buttons. The teleop_twist_joy (https://github.com/ros2/teleop_twist_joy/tree/humble/config) package contains configurations for different types of joysticks. The PDP joystick was not supported th it's necessary to create a *pdp.config.yaml* file which is copied below:

```
teleop_twist_joy_node:
  ros__parameters:
    axis_linear:  # Left thumb stick vertical
      x: 1
    scale_linear:
      x: 0.7
    scale_linear_turbo:
      x: 1.5

    axis_angular:  # Left thumb stick horizontal
      yaw: 0
    scale_angular:
      yaw: 0.4

    enable_button: 4  # Left 'LB' button
    enable_turbo_button: 5  # Right 'RB' button
```

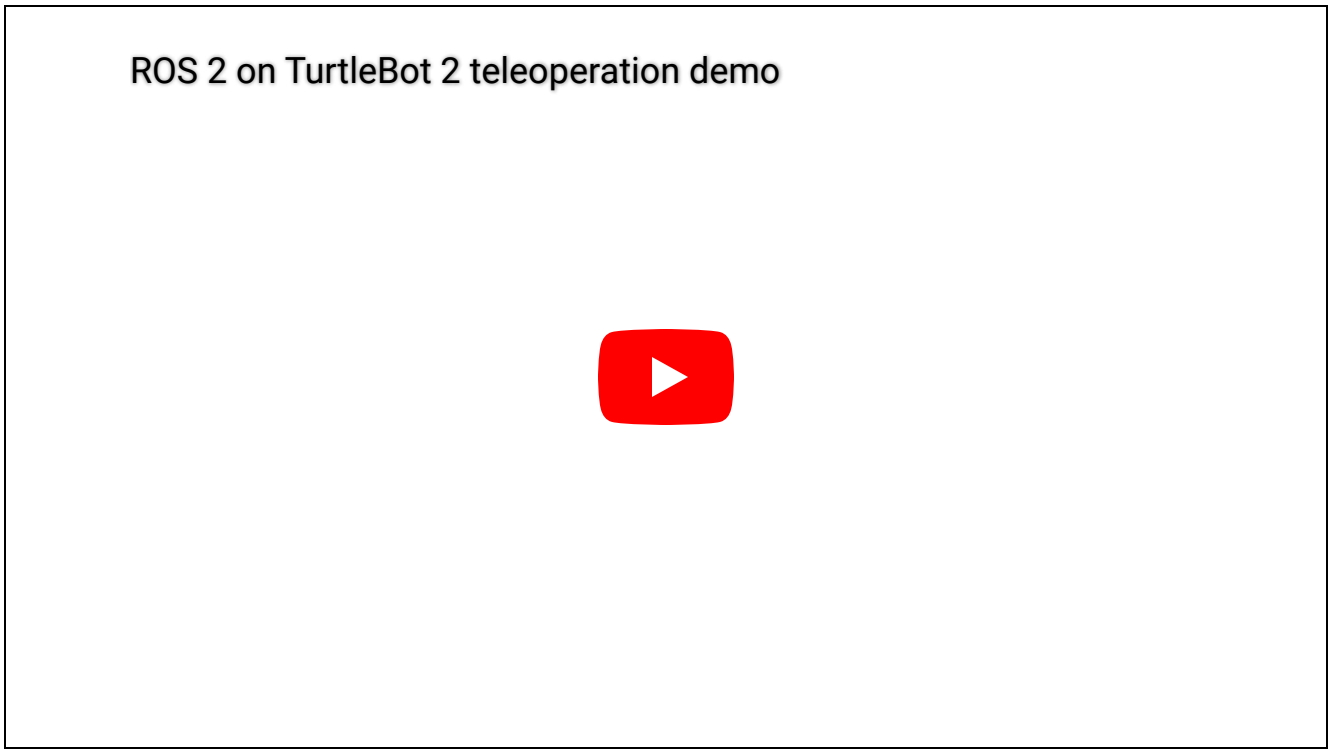
To control the robot via the joystick, run the following command in terminal: `ros2 launch teleop_twist_joy teleop-launch.py config_filepath:='/path/to/ros2_ws/src/configs/pdp.config.yaml' joy_vel:='commands/velocity'`.

Move the robot using the left stick whilst pressing the left *LB* button. To go faster (turbo mode) press th right *RB* button. The *teleop-launch.py* launch file also runs joy_node so there is no need to run it separately.

[Update] Submitted a PR to ROS Rolling to add native support for PDP joystick controllers (https://github.com/ros2/teleop_twist_joy/pull/41). For future ROS 2 distributions, the joystick can be launched using the simplified command: `ros2 launch teleop_twist_joy teleop-launch.py joy_config:='pdp' joy_vel:='commands/velocity'`.

Test teleoperation functionality from a workstation as well as when physically connected to the robot t ensure that both scenarios work well for debugging purposes.

The video below shows the Kobuki TurtleBot navigating using ROS 2 teleoperation.



Autonomy

This article demonstrates how to control a TurtleBot 2 via ROS 2 Humble using both keyboard and joystick teleoperation. Teleoperation is the baseline upon which autonomy is built. Autonomy, which is the ultimate goal of robotics, can be divided into two categories: shared autonomy and full autonomy.

Shared autonomy (or shared control) involves the intelligent combination of human and robot interactions to aid control. This is typical in flight control software for drones, where the pilot remains i control of the flight but benefits from software compensation for uncontrolled movements or the wind order to fly smoothly.

Full autonomy refers to a robot running and completing tasks without human intervention. Achieving full autonomy is hard. Most robots today, even with cutting-edge technology, require some level of human intervention. The goal is to reduce the number of critical interventions required whilst increasir the period of continuous fully autonomous operation.

Future articles will discuss integrating a stereo camera with the TurtleBot 2 platform and the algorithm available for autonomous vision-based navigation with ROS 2.