02. Linux, ROS introduction

Lecture

Linux principles

- (Was) the only OS supported by ROS
- Security
- Efficieny
- Open-source
- Community support
- User freedom
- Distributions: **Ubuntu**, Linux Mint, Debian, etc.
- Terminal usage more dominant

Suggestion

Install **Terminator** terminal emulator:

```
sudo apt update
sudo apt install terminator
```

Linux commands

See some basic commands below:

- Run as administrator with `sudo`
- Manual of command `man`, e.g. `man cp`
- Package management `apt`, e.g. `apt update`, `apt install`
- Navigation `cd`
- List directory contents `ls`
- Create file `touch`
- Copy file `cp`
- Move file `mv`
- Remove file `rm`
- Make directory `mkdir`
- Remove directory `rmdir`
- Make a file executable `chmod +x <filename>`
- Safe restart: Crtl + Alt + PrtScr + REISUB
- If not sure, just google the command

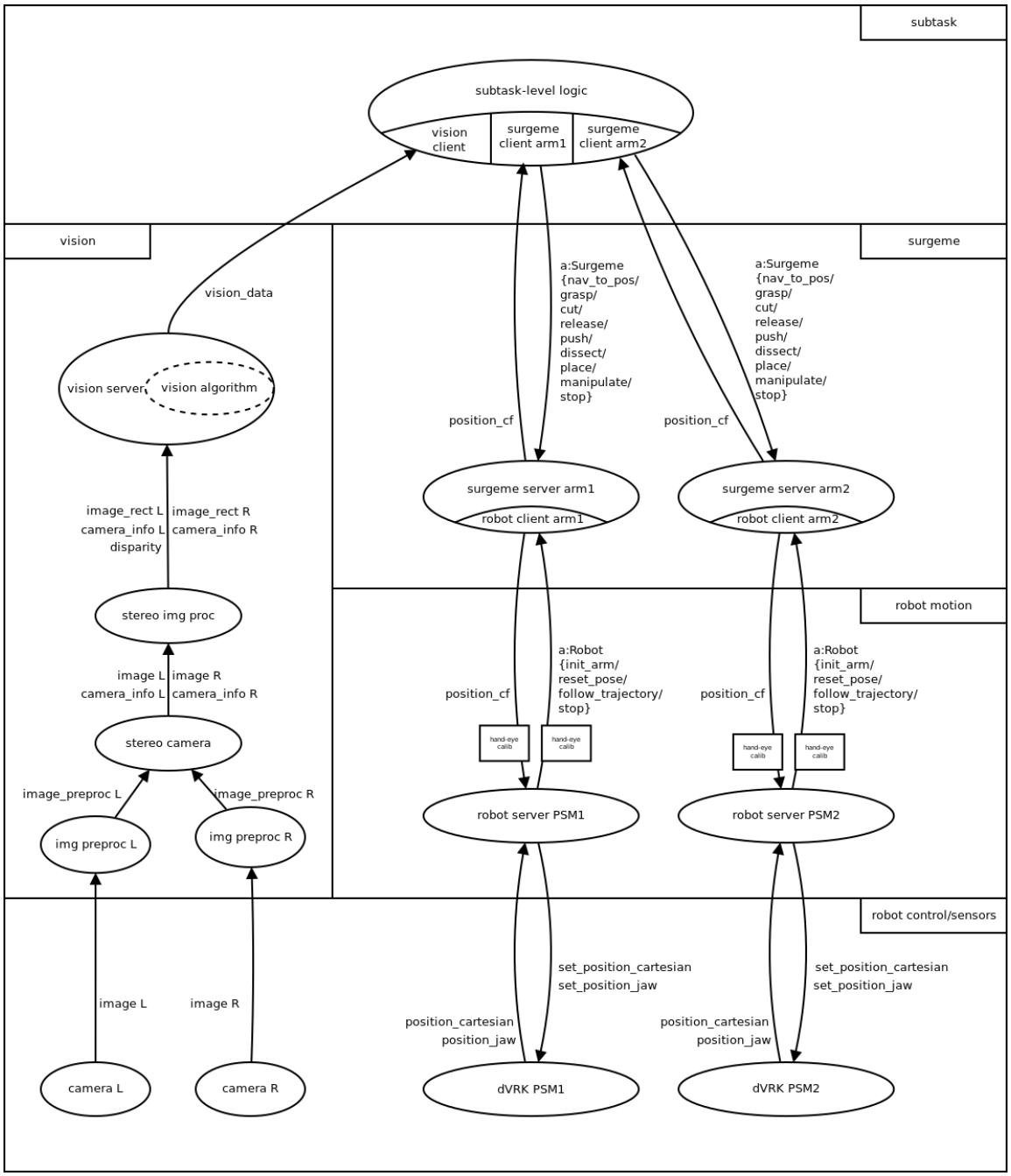
ROS 1 → ROS 2

- ROS 2 was rewritten from scratch
- More modular architecture
- Improved support for real-time systems
- Support for multiple communication protocols
- Better interoperability with other robotic systems
- Focus on standardization and industry collaboration
- No ROS Master
- No Devel space
- `rclpy`, `rclcpp`
- More structured code (`Node` class)

- Different build system
- Platforms: Windows, OS X, Linux

ROS principles

ROS node

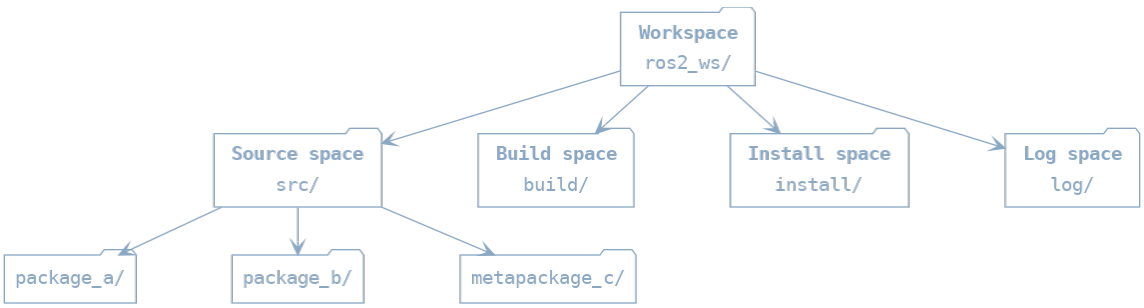


- Executable part of ROS:
    - python scripts
    - compiled C++ code
- A process that performs computation
- Inter-node communication:
    - ROS topics (streams)
    - ROS parameter server
    - Remote Procedure Calls (RPC)

    - ROS services
    - ROS actions
- Meant to operate at a fine-grained scale
- Typically, a robot control system consists of many nodes, like:
    - Trajectory planning
    - Localization
    - Read sensory data
    - Process sensory data
    - Motor control
    - User interface
    - etc.

ROS workspace

> **Colcon workspace**
>
> A folder where packages are modified, built, and installed.



- Source space:
    - Source code of colcon packages
    - Space where you can extract/checkout/clone source code for the packages you want to build
- Build space
    - Colcon is invoked here to build packages
    - Colcon and CMake keep intermediate files here

- Install space:
  - Each package will be installed here; by default each package will be installed into a separate subdirectory
- Log space:
  - Contains various logging information about each colcon invocation

ROS package

> **ROS package principle**
>
> Enough functionality to be useful, but not too much that the package is heavyweight and difficult to use from other software.

- Main unit to organize software in ROS
- Buildable and redistributable unit of ROS code
- Consists of (in the case of Python packages):
  - `package.xml` file containing meta information about the package
    - name
    - version
    - description
    - dependencies
    - etc.
  - `setup.py` containing instructions for how to install the package
  - `setup.cfg` is required when a package has executables, so ros2 run can find them
  - `/<package_name>` - a directory with the same name as your package, used by ROS 2 tools to find your package, contains `__init__.py`
  - Anything else
- `ros2 run turtlesim turtlesim_node`

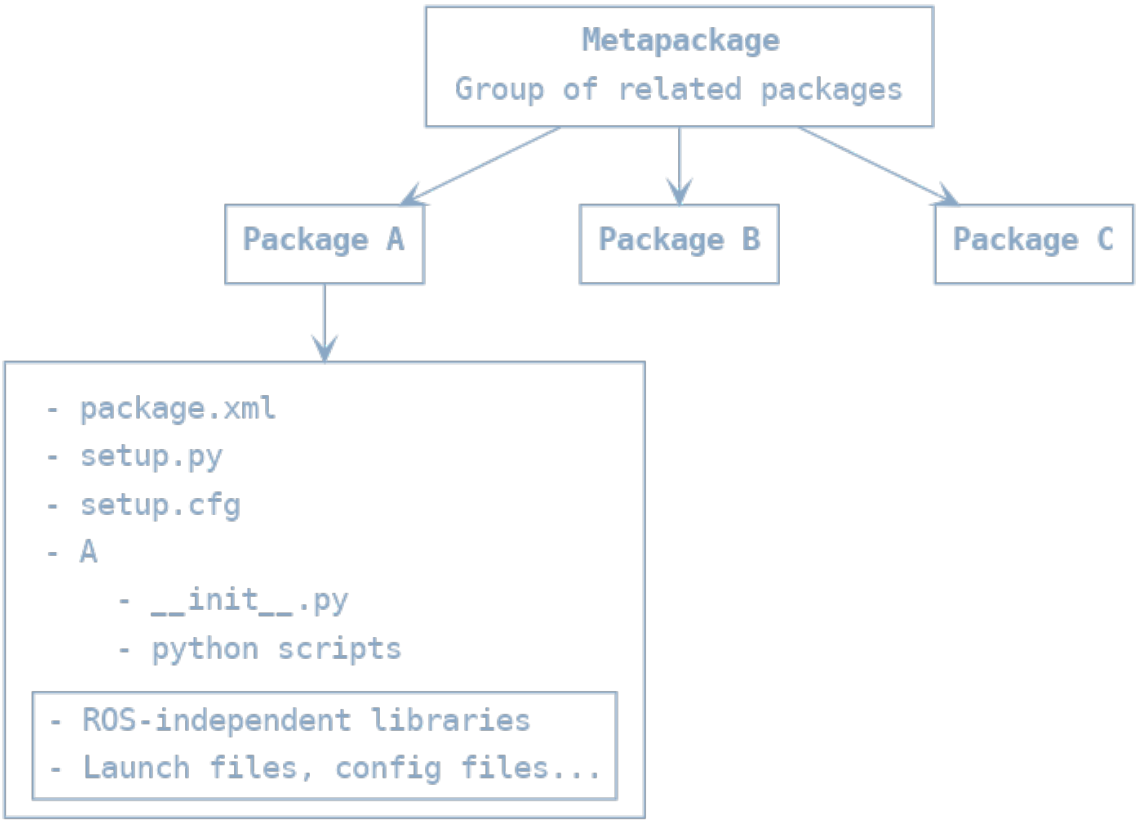
ROS dependencies

After cloning a new package, use the following command to install depenencies:

```
rosdep install --from-paths src --ignore-src -r -y
```

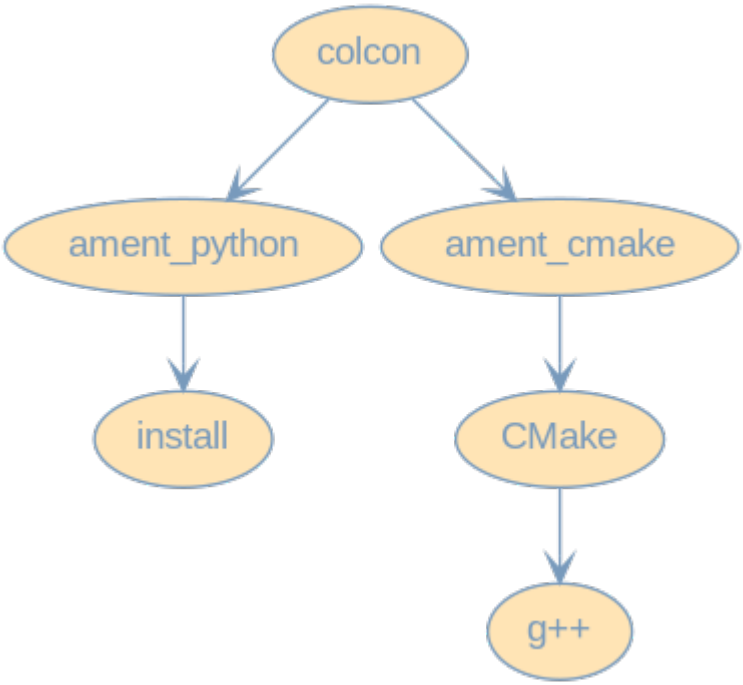
CMake

For CMake packages (C++), the package contents will be different.



ROS build system---Colcon

- System for building software packages in ROS



Environmental setup file

- setup.bash
- generated during init process of a new workspace
- extends shell environment
- ROS can find any resources that have been installed or built to that location

```
source ~/ros2_ws/install/setup.bash
```

Practice

1: Turtlesim

1. Start `turtlesim_node` and `turtle_teleop_key` nodes with the following commands, in separate terminal windows:

   ```
   ros2 run turtlesim turtlesim_node
   ```

```
ros2 run turtlesim turtle_teleop_key
```

> **Tip**
>
> In **Terminator**, you can further divide the given window with Ctrl-Shift-O, Ctrl-Shift-E key combinations. Ctrl-Shift-W closes the active window.

> **Abort execution**
>
> `Ctrl-C`

2. Running the following ROS commands can provide useful information:

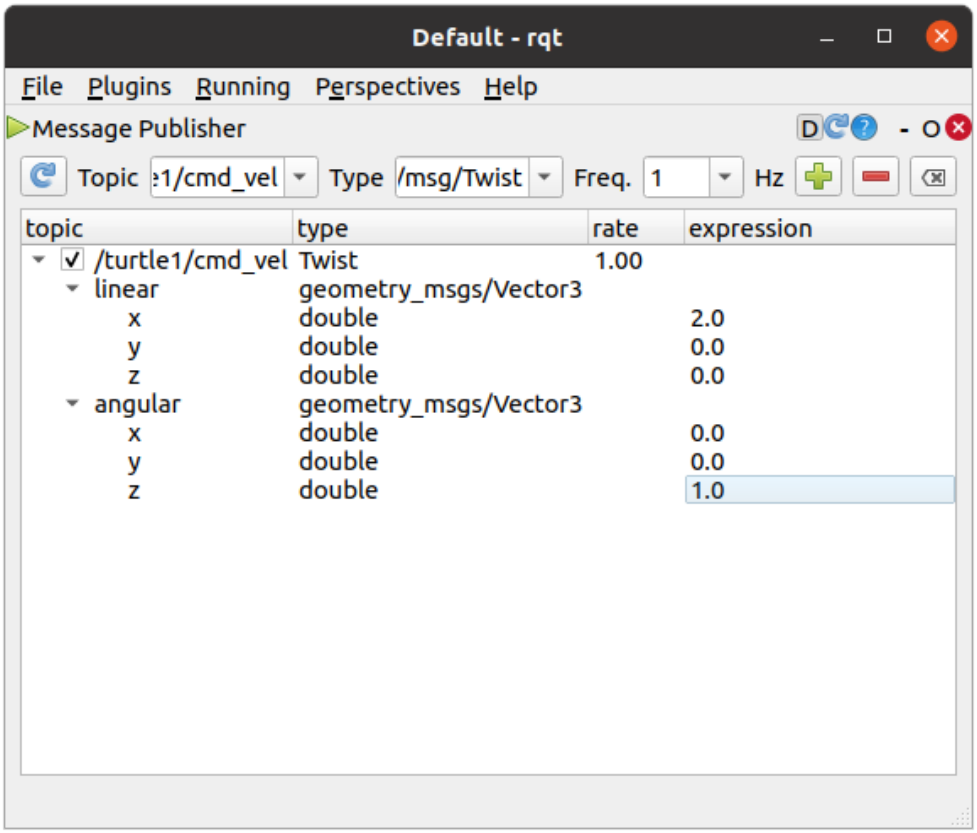
```
ros2 wtf
ros2 node list
ros2 node info /turtlesim
ros2 topic list
ros2 topic info /turtle1/cmd_vel
ros2 interface show geometry_msgs/msg/Twist
ros2 topic echo /turtle1/cmd_vel
```

3. Start `rqt_gui` with the following command:

```
ros2 run rqt_gui rqt_gui
```

4. Display the running nodes and topics in `rqt_gui`: Plugins → Introspection → Node Graph.

5. Publish to the `/turtle1/cmd_vel` topic also using `rqt_gui`: Plugins → Topics → Message Publisher.



2: ROS 2 workspace creation

1. Let's create a new ROS2 workspace with the name `ros2_ws`.

```
mkdir -p ~/ros2_ws/src
```

3: ROS 2 package creation

1. Let's create a new ROS2 package with the name `ros2_course` and a Hello World.

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python --node-name hello ros2_course
```

Syntax

`ros2 pkg create --build-type ament_python <package_name>`

2. Build the workspace.

```
cd ~/ros2_ws
colcon build --symlink-install
```

Symlink

The option `--symlink-install` links the source scripts to the Install space, so we don't have to build again after modification.

3. Insert the following line at the end of the `~/.bashrc` file:

```
source ~/ros2_ws/install/setup.bash
```

Import to QtCreator

New file or project → Other project → ROS Workspace. Select Colcon as Build System and `ros2_ws` as Workspace path.

Import to CLion

Set the Python interpreter to Python 3.8, `/usr/bin/python3`. Add the follwong path: `/opt/ros/foxy/lib/python3.8/site-packages`. Create the file `compile_commands.json` in the directory `~/ros2_ws/build` with the following content:

```
[
]
```

4. Test Hello World:

```
ros2 run ros2_course hello
```

4: Implementing a Publisher in Python

1. Navigate to the `ros2_ws/src/ros2_course/ros2_course` folder and create the `talker.py` file with the content below.

```
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class Talker(Node):

    def __init__(self):
        super().__init__('talker')
        self.publisher_ = self.create_publisher(String, 'chatter', 10)
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0

    def timer_callback(self):
        msg = String()
        msg.data = 'Hello World: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1


def main(args=None):
    rclpy.init(args=args)
    talker_node = Talker()
    rclpy.spin(talker_node)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    talker_node.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

2. Add a new entry point in the `setup.py` file:

```
'talker = ros2_course.talker:main',
```

1. Build and run the node:

```
cd ~/ros2_ws
colcon build --symlink-install
ros2 run ros2_course talker
```

2. Check the output of the node using `ros2 topic echo` command or `rqt_gui`.

5: Implementing a Subscriber in Python

1. Navigate to the `ros2_ws/src/ros2_course/ros2_course` folder and create the `listener.py` file with the content below.

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String


class Listener(Node):

    def __init__(self):
        super().__init__('listener')
        self.subscription = self.create_subscription(
            String,
            'chatter',
            self.listener_callback,
            10)
        self.subscription  # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('I heard msg: "%s"' % msg.data)


def main(args=None):
    rclpy.init(args=args)
    listener_node = Listener()
    rclpy.spin(listener_node)

    # Destroy the node explicitly
    # (optional - otherwise it will be done automatically
    # when the garbage collector destroys the node object)
    listener_node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

2. Add a new entry point in the `setup.py` file:

```
'listener = ros2_course.listener:main',
```

3. Build and run both nodes:

```
cd ~/ros2_ws
colcon build --symlink-install
ros2 run ros2_course talker
```

```
ros2 run ros2_course listener
```

1. Use `rqt_gui` to display the nodes and topics of the running system:

```
ros2 run rqt_gui rqt_gui
```

Useful links

- ROS 2 Tutorials
- What is a ROS 2 package?