CSC398 Autonomous Robots
- Introduction into ROS 2 (2) -

Ubbo Visser

Department of Computer Science
College of Arts and Sciences
University of Miami

September 2026

UNIVERSITY OF MIAMI
ROBOCANES

OVERVIEW

- ROS 2 package structure
- ROS C++ client library (rclcpp)
- ROS 2 subscribers and publishers
- ROS 2 parameters
- RViz2 visualization

ROS PACKAGES

- ROS 2 software is organized into **packages**, which can contain source code, launch files, configuration files, interface definitions, data, and documentation.
- A package that uses functionality or interfaces from other packages declares these as **dependencies** in package.xml.

Plain text

```
package_name/
├── config/
│   └── *.yaml
├── include/package_name/
├── launch/
│   └── *.launch.py
├── src/
├── test/
├── CMakeLists.txt
└── package.xml
```

Plain text

```
package_name_interfaces/
├── action/
│   └── *.action
├── msg/
│   └── *.msg
├── srv/
│   └── *.srv
├── CMakeLists.txt
└── package.xml
```

Separate interface definition packages!

Creating a package:

```
~$ros2 pkg create package_name --build-type ament_cmake --dependencies ...
```

Details at: First ROS2 package

ROS 2 PACKAGES

- The package.xml file hosts and defines the properties of a package
  - Name of the package
  - Versioning
  - Authors
  - Dependencies
  - …

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" sche
<package format="3">
  <name>assignment2-uv-cpp</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="robocanes@todo.todo">robocanes</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>rclcpp</depend>
  <depend>sensor_msgs</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

```
In [ROS_WS]/src:
ros2 pkg create assignment2-uv-cpp --build-type ament_cmake --dependencies rclcpp sensor_msgs
```

Details at:
https://docs.ros.org/en/humble/How-To-Guides/Developing-a-ROS-2-Package.html

ROS 2 PACKAGES

- The CMakeLists.txt file defines the necessary inputs for the build system
  - Required CMake version
  - Package/project name
  - Find ROS 2 and other CMake packages needed for build (find_package())
  - Define libraries/executables (add_library(), add_executable())
  - Specify ROS dependencies (ament_target_dependencies())
  - Tests to build (ament_add_gtest())
  - Install rules (install())
  - Register package with ament (ament_package())

Details at:
https://docs.ros.org/en/humble/How-To-Guides/Ament-CMake-Documentation.html

```
cmake_minimum_required(VERSION 3.8)
project(assignment2-uv-cpp)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(sensor_msgs REQUIRED)

add_executable(assignment2-uv-cpp src/assignment2-uv-cpp.cpp)

ament_target_dependencies(
  assignment2-uv-cpp
  rclcpp
  sensor_msgs)

install(TARGETS
  assignment2-uv-cpp
  DESTINATION lib/${PROJECT_NAME}
)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  # the following line skips the linter which checks for copyrights
  # comment the line when a copyright and license is added to all source f
  set(ament_cmake_copyright_FOUND TRUE)
  # the following line skips cpplint (only works in a git repo)
  # comment the line when this package is in a git repo and when
  # a copyright and license is added to all source files
  set(ament_cmake_cpplint_FOUND TRUE)
  ament_lint_auto_find_test_dependencies()
endif()

ament_package()
```

ROS 2 PACKAGES

- The package.xml file hosts and defines the properties of a package
  - Name of the package
  - Versioning
  - Authors
  - Dependencies
  - …

```xml
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" sch
<package format="3">
  <name>assignment2-uv-py</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="robocanes@todo.todo">robocanes</maintainer>
  <license>TODO: License declaration</license>

  <depend>rclpy</depend>
  <depend>sensor_msgs</depend>

  <test_depend>ament_copyright</test_depend>
  <test_depend>ament_flake8</test_depend>
  <test_depend>ament_pep257</test_depend>
  <test_depend>python3-pytest</test_depend>

  <export>
    <build_type>ament_python</build_type>
  </export>
</package>
```

```
In [ROS_WS]/src:
ros2 pkg create assignment2-uv-py --build-type ament_python --dependencies rclpy sensor_msgs
```

Details at:
https://docs.ros.org/en/humble/How-To-Guides/Developing-a-ROS-2-Package.html

ROS 2 PACKAGES

- The **setup.py** file defines the necessary inputs for the build system
  - Python packages to install (packages)
  - Package data/resources (data_files)
  - Python dependencies (install_requires)
  - Package metadata (maintainer, description, license)
  - Executable ROS 2 nodes (entry_points)
  - Register package with the ROS 2 package index

```python
from setuptools import find_packages, setup

package_name = 'assignment2-uv-py'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='robocanes',
    maintainer_email='robocanes@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    extras_require={
        'test': [
            'pytest',
        ],
    },
    entry_points={
        'console_scripts': [
        ],
    },
)
```

Details at:
https://docs.ros.org/en/humble/How-To-Guides/Developing-a-ROS-2-Package.html

ROS 2 C++ CLIENT LIBRARY (RCLCPP)

```cpp
#include <memory>
#include "rclcpp/rclcpp.hpp"

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);

  auto node = rclcpp::Node::make_shared("hello_world");

  rclcpp::Rate loop_rate(10);

  unsigned int count = 0;

  while (rclcpp::ok())
  {
    RCLCPP_INFO(node->get_logger(),
                "Hello World %u", count);

    rclcpp::spin_some(node);

    loop_rate.sleep();
    count++;
  }

  rclcpp::shutdown();
  return 0;
}
```

- ROS 2 C++ client library header: #include "rclcpp/rclcpp.hpp"
- rclcpp::init(...) must be called before using ROS 2 functionality
- A rclcpp::Node object represents the ROS 2 node and provides access to topics, services, parameters, timers, etc.
- rclcpp::Rate is a helper class for running loops at a desired frequency
- rclcpp::ok() checks whether ROS 2 should continue running
- RCLCPP_INFO() writes a log message using the node's logger
- rclcpp::spin_some(node) processes available callbacks
- rclcpp::shutdown() shuts down the ROS 2 context

Details at:
https://docs.ros.org/en/humble/p/rclcpp

ROS 2C++ CLIENT LIBRARY (RCLCPP) — NODES AND NAMESPACES

- In ROS 2, communication interfaces are created directly from a rclcpp::Node
- A node has a **name** and may have a **namespace**

```
auto node = rclcpp::Node::make_shared("hello_world");
```

```
auto node = rclcpp::Node::make_shared("hello_world", "/namespace");
```

- For a **node** in **namespace** looking up **topic**, these will resolve to:

```
/namespace/topic                #relative
/namespace/my_node/topic        #private
/topic                          #absolute
```

Details at:
https://design.ros2.org/articles/topic_and_service_names.html

ROS 2 PYTHON CLIENT LIBRARY (RCLPY) — LOGGING

- Mechanism for logging human-readable messages from nodes
- Instead of print(), use the node's ROS 2 logger
- Different severity levels: Debug, Info, Warn, Error, Fatal
- Logger is obtained from the node:

```
self.get_logger().debug('Debug message')
self.get_logger().info('Information message')
self.get_logger().warning('Warning message')
self.get_logger().error('Error message')
self.get_logger().fatal('Fatal message')
```

	Typical use
Debug	Detailed diagnostic information
Info	Normal operational information
Warn	Unexpected but recoverable condition
Error	Operation failed
Fatal	Serious/unrecoverable condition

To see output in terminal:
Set logging level

```
Node(
    package='my_package',
    executable='listener',
    output='screen'
)
```

```
~/$ ros2 run my_package my_node --ros-args --log-level debug
```

Details at: https://docs.ros.org/en/humble/Concepts/Intermediate/About-Logging.html

ROS 2 C++ CLIENT LIBRARY (RCLCPP) — SUBSCRIBER

- Start listening to a topic by calling create_subscription() on the node

```
subscription_ = this->create_subscription<std_msgs::msg::String>(
    "chatter", 10, callback);
```

- When a message is received, the callback function is called with the message as an argument
- Keep the subscription object alive as long as you want to receive messages
- rclcpp::**spin**() processes callbacks and blocks until the node is shut down

```
#include <memory>
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

void callback(const std_msgs::msg::String::SharedPtr msg)
{
  RCLCPP_INFO(rclcpp::get_logger("listener"),
              "I heard: '%s'", msg->data.c_str());
}

int main(int argc, char * argv[])
{

  rclcpp::init(argc, argv);

  auto node = rclcpp::Node::make_shared("listener");
  auto subscriber =
    node->create_subscription<std_msgs::msg::String>(
      "chatter", 10, callback);

  rclcpp::spin(node);
  rclcpp::shutdown();

  return 0;
}
```

Details at:
https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html

ROS 2 C++ CLIENT LIBRARY (RCLCPP) — PUBLISHER

- Create a publisher from the node

```
auto publisher =
  node->create_publisher<std_msgs::msg::String>(
    "chatter", 10);
```

- Create the message contents
- Publish content with

```
publisher->publish(message);
```

```
#include <memory>
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);

  auto node = rclcpp::Node::make_shared("talker");
  auto publisher =
    node->create_publisher<std_msgs::msg::String>(
      "chatter", 10);
  rclcpp::Rate loop_rate(10);
  unsigned int count = 0;

  while (rclcpp::ok())
  {
    std_msgs::msg::String message;
    message.data = "Hello World " + std::to_string(count);

    RCLCPP_INFO(node->get_logger(),
                "%s", message.data.c_str());
    publisher->publish(message);
    loop_rate.sleep();
    count++;
  }
  rclcpp::shutdown();

  return 0;
}
```

Details at:
https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html

ROS 2 C++ CLIENT LIBRARY (RCLCPP) — OOP

```cpp
#include "rclcpp/rclcpp.hpp"
#include "my_package/MyPackage.hpp"

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);

  auto node = std::make_shared<my_package::MyPackage>();

  rclcpp::spin(node);

  rclcpp::shutdown();
  return 0;
}
```

MyPackage.hpp / MyPackage.cpp

Main node class, providing the ROS 2 interface:

- subscribers
- publishers
- parameters
- timers
- services, etc.

```cpp
class MyPackage : public rclcpp::Node
```

Algorithm.hpp / Algorithm.cpp

Class implementing the algorithmic part of the node.

- The algorithmic code can remain ROS-independent.

```cpp
class Algorithm
```

Register a class method as subscriber callback

```cpp
subscription_ = this->create_subscription<std_msgs::msg::String>(
    "chatter", 10,
    std::bind(&MyPackage::callback, this, _1));
```

```cpp
C++
void MyPackage::callback(
    const std_msgs::msg::String::SharedPtr msg)
{
    // process message
}
```

Details at:
https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html

ROS 2 PYTHON CLIENT LIBRARY (RCLPY)

```python
import random

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

def getName():
    names = ['John', 'Doe', 'Jane', 'Smith', 'Alice']
    return random.choice(names)

def getStatus():
    status = [
        'happy', 'sad', 'angry', 'excited',
        'bored', 'tired', 'hungry', 'awake'
    ]
    return random.choice(status)

def main(args=None):
    rclpy.init(args=args)
    node = Node('publisher_node')
    publisher = node.create_publisher(String, '/ubbo1', 1)

    def callback():
        msg = String()
        msg.data = '%s is %s' % (getName(), getStatus())
        publisher.publish(msg)
        node.get_logger().info(msg.data)

    node.create_timer(1.0, callback)

    try:
        rclpy.spin(node)
    finally:
        node.destroy_node()
        rclpy.shutdown()

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

- ROS 2 Python client library: import rclpy
- rclpy.init() must be called before using ROS 2 functionality
- Node(...) creates a ROS 2 node
- node.create_publisher(...) creates a publisher for a topic
- node.get_logger().info(...) logs messages
- node.create_timer(...) invokes a callback periodically
- rclpy.spin(node) processes callbacks until the node is shut down
- rclpy.shutdown() shuts down ROS 2

Details at:
Publisher/Subscriber python

ROS 2 PYTHON CLIENT LIBRARY (RCLPY) — LOGGING

- Mechanism for logging human-readable text from nodes
- Instead of print(), use e.g. node.get_logger().info()
- Logging to the console and /rosout
- Different severity levels: Debug, Info, Warn, Error, Fatal
-
```
node.get_logger().debug('Debug message')
node.get_logger().info('Info message')
node.get_logger().warning('Warning message')
node.get_logger().error('Error message')
node.get_logger().fatal('Fatal message')
```
- Logging level can be configured at runtime
- Supports additional features such as throttled and once-only logging

	Typical use
Debug	Detailed diagnostic information
Info	Normal operational information
Warn	Unexpected but recoverable condition
Error	Operation failed
Fatal	Serious/unrecoverable condition

To see output in terminal:
Set logging level from the command line:

```
ros2 run my_package my_node \
  --ros-args --log-level debug
```

Details at:
https://docs.ros.org/en/humble/Concepts/Intermediate/About-Logging.html

ROS 2 PYTHON CLIENT LIBRARY (RCLPY) — SUBSCRIBER

- Start listening to a topic by calling create_subscription() on the node

```
subscriber = node.create_subscription(
    LaserScan, '/hsrb/base_scan', callback, 10)
```

- When a message is received, the callback function is called with the contents of the message as the argument
- Hold on to the subscriber object until you want to unsubscribe
- **rclpy.spin(node)** processes callbacks and blocks until the node is shut down

```python
import math
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan

class LaserScanListener(Node):
    """Log the minimum valid distance in each received laser scan."""

    def __init__(self):
        """Create the laser scan subscription."""
        super().__init__('laser_scan_listener')
        self.subscription = self.create_subscription(
            LaserScan,
            '/hsrb/base_scan',
            self.scan_callback,
            10,
        )

    def scan_callback(self, scan):
        """Log the closest finite range in a laser scan."""
        valid_ranges = [distance for distance in scan.ranges
                        if math.isfinite(distance)]

        if valid_ranges:
            min_distance = min(valid_ranges)
            self.get_logger().info(
                f'Minimum distance to object: {min_distance:.2f} meters')
        else:
            self.get_logger().warning('No range data available.')

def main(args=None):
    """Run the laser scan listener node."""
    rclpy.init(args=args)
    node = LaserScanListener()

    try:
        rclpy.spin(node)
    finally:
        node.destroy_node()
        rclpy.shutdown()

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

Details at:
https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html

ROS 2 PYTHON CLIENT LIBRARY (RCLPY) — PUBLISHER

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

def getName():
    names = ['John', 'Doe', 'Jane', 'Smith', 'Alice']
    return random.choice(names)

def getStatus():
    status = [
        'happy',
        'sad',
        'angry',
        'excited',
        'bored',
        'tired',
        'hungry',
        'awake',
    ]
    return random.choice(status)

class PublisherNode(Node):
    def __init__(self):
        super().__init__('publisher_node')
        self.publisher = self.create_publisher(String, '/ubbo1', 1)
        self.timer = self.create_timer(1.0, self.callback)

    def callback(self):
        msg = String()
        msg.data = '%s is %s' % (getName(), getStatus())
        self.publisher.publish(msg)
        self.get_logger().info(msg.data)

def main(args=None):
    rclpy.init(args=args)
    node = PublisherNode()

    try:
        rclpy.spin(node)
    finally:
        node.destroy_node()
        rclpy.shutdown()

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

- Create a publisher

```python
self.publisher = self.create_publisher(String, '/ubbo1', 1)
```

- Create the message contents
- Publish content with

```python
self.publisher.publish(msg)
```

Details at:
Publisher/Subscriber python

ROS 2 PARAMETERS

- Parameters are associated with individual nodes
- Nodes use parameters to store and retrieve configuration values at runtime
- Best used for relatively static configuration data
- Parameters can be set through:
  - command line
  - launch files
  - YAML parameter files

List parameters with

```
~$ros2 param list
```

Get values

```
~$ros2 param get /node_name parameter_name
```

Set values

```
~$ros2 param set /node_name parameter_name value
```

config.yaml

```yaml
camera:
  ros__parameters:
    left:
      name: left_camera
      exposure: 1.0
    right:
      name: right_camera
      exposure: 1.1
```

I.e: these parameters apply to the node named camera

Python launch file

```python
Node(
    package='my_package',
    executable='my_node',
    name='camera',
    parameters=['config/config.yaml']
)
```

Details at:
Humble parameters

ROS 2 PARAMETERS — C++ API

- Declare a parameter in C++:

```
node->declare_parameter<std::string>("topic", "chatter");
```

- Get the parameter value:

```
std::string topic;
node->get_parameter("topic", topic);
```

get_parameter() returns true if the parameter exists and has the expected type; false otherwise

- Parameters belong to the node, rather than to a global parameter server
- Parameter names can use hierarchical notation:

```
node->declare_parameter<double>(
    "camera.left.exposure", 1.0);
```

```
double exposure;
node->get_parameter(
    "camera.left.exposure", exposure);
```

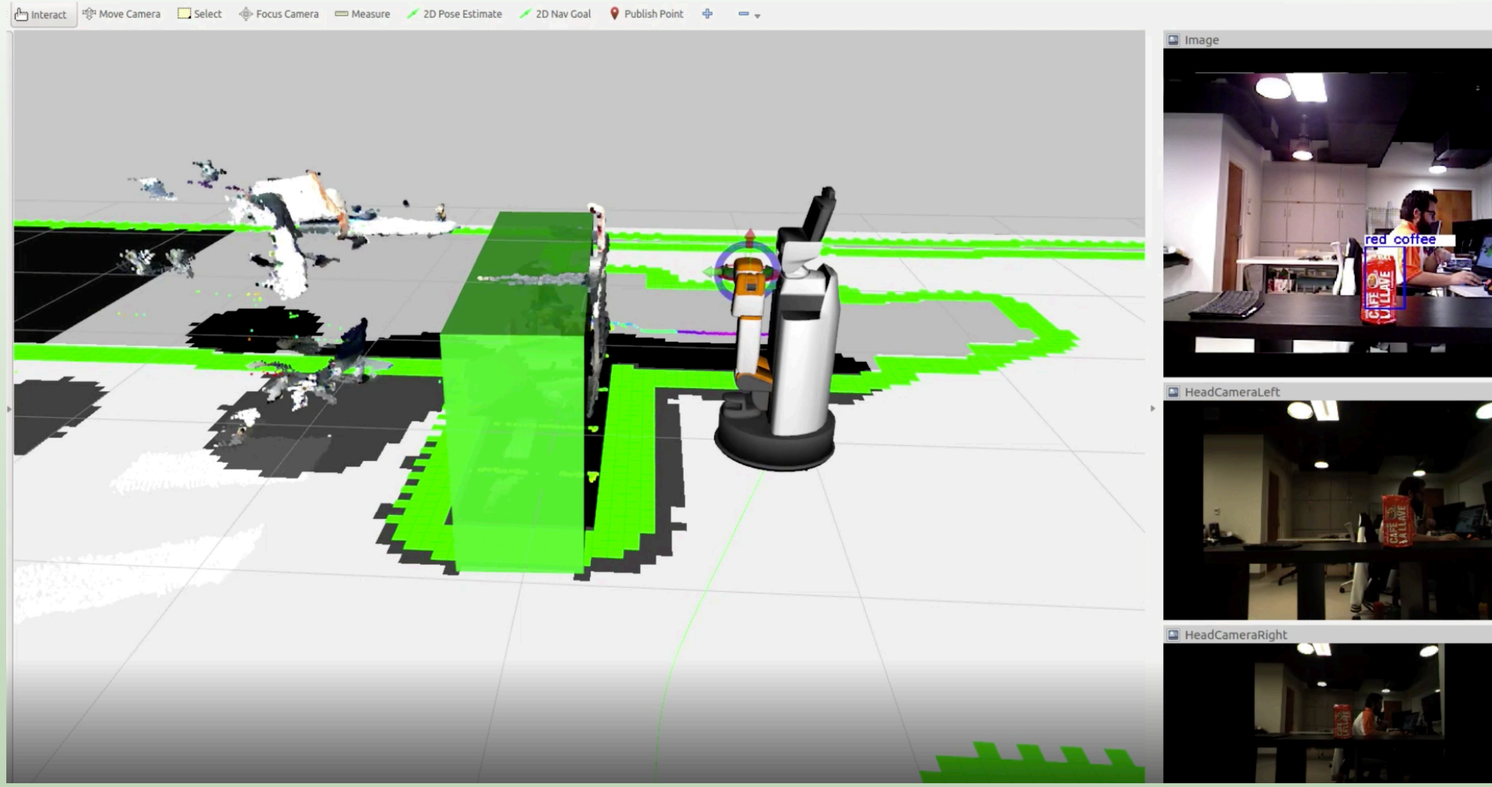
```
auto node = rclcpp::Node::make_shared("camera");

node->declare_parameter<std::string>(
    "topic", "chatter");

std::string topic;

if (!node->get_parameter("topic", topic))
{
  RCLCPP_ERROR(node->get_logger(),
               "Could not get topic parameter!");
}
```

Declaration: ROS 2 nodes normally declare parameters before using them.

Details at:
Using parameters in a class (C++)

RVIZ2

- 3D visualization tool for ROS 2
- Subscribes to topics and visualizes message contents
- Different camera views (orthographic, top-down, etc.)
- Interactive tools for publishing user input
- Save and load setup as RViz2 configuration (.rviz)
- Extensible with plugins
- Supports ROS 2 QoS settings for displays



Run RViz2:

```
~$rviz2
```

Details at:
RViz2

FURTHER REFERENCES

- ROS 2 Documentation
  - ROS 2 Humble Documentation
  - ROS 2 Concepts
- Installation
  - ROS 2 Humble Installation
- Tutorials
  - ROS 2 Tutorials
- Packages
  - ROS Package Index
- ROS 2 CLI
  - ROS 2 CLI documentation
- ROS2 C++ API
  - rclcpp documentation
- ROS 2 Python API
  - rclpy documentation
- ROS 2 Design
  - ROS 2 Design documents