Physical Interaction for Aerial and Space Robots (AE4324)

Tutorial 01:

What is ROS 2?

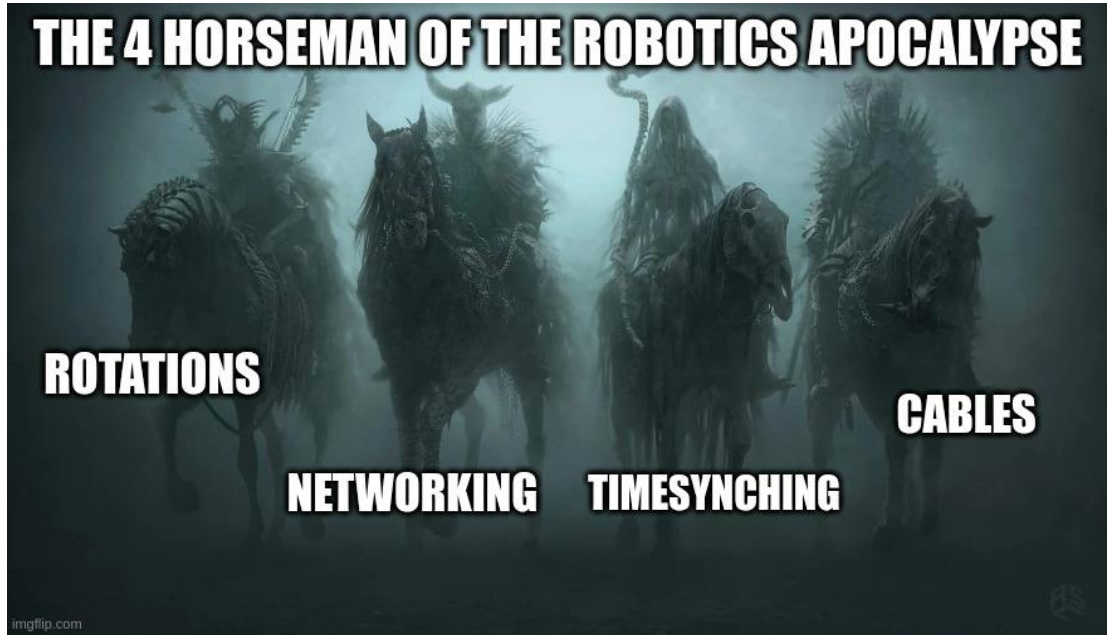
TU Delft

BIOMORP MORPHIC PHIC LAB

Tutorial 01:
What is ROS 2 ?

The Role of ROS in Robotics

- Before ROS:



ROS
Robot Operating System

BIOMORP
MORPHIC
PHICLAB

The Role of ROS in Robotics

- After ROS:

ROS
Robot Operating System

BIOMORP
MORPHIC
PHICLAB

- ROS as a middleware that takes care of lot's of commonly occuring tasks
  - Communication
  - Visualization
  - Logging
  - ...

ROS vs ROS2

ROS

3. List of Distributions

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			June 27, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019

ROS 2

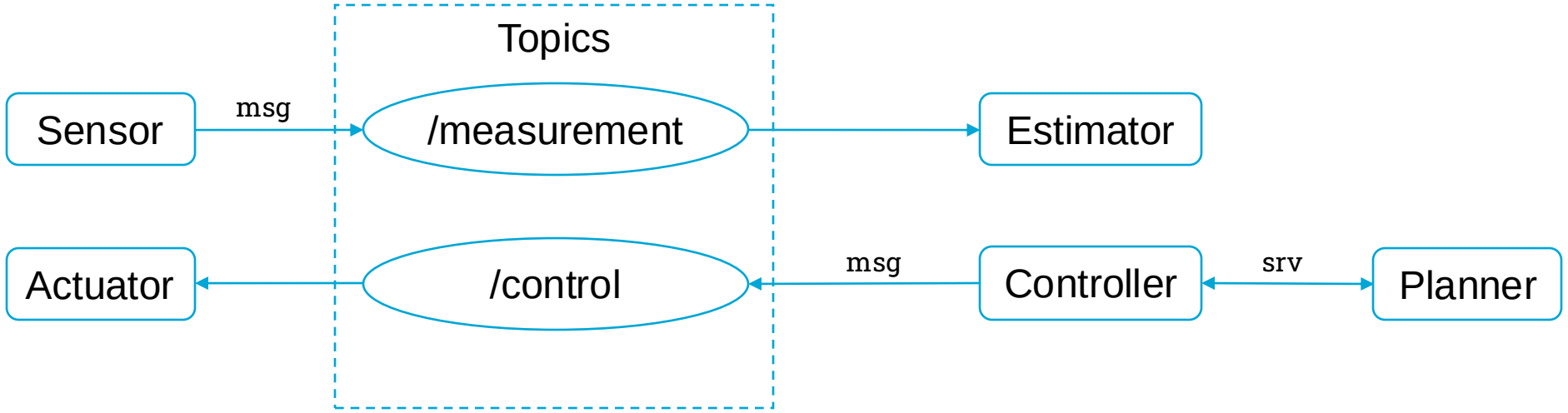
List of Distributions

Below is a list of current and historic ROS 2 distributions. Rows in the table marked in green are the currently supported distributions.

Distro	Release date	Logo	EOL date	ROS Boss
Jazzy Jalisco	May 23, 2024		May 2029	Marco A. Gutiérrez
Iron Irwini	May 23, 2023		December 4, 2024	Yadunund Vijay
Humble Hawksbill	May 23, 2022		May 2027	Audrow Nash
Galactic Geochelone	May 23, 2021		December 9, 2022	Scott Logan

Overview

- In a ROS application individual **Nodes**
  - pass **messages** over **topics** in a "publisher-subscriber" architecture
  - can be configured using **paramters**
  - invoke **services** in a "client-agent" architecture
- Example:



Simple Publisher <=> Subscriber C++

BIOMORP
MORPHIC
PHICLAB

```cpp
#include <chrono>
#include <memory>
#include <string>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

using namespace std::chrono_literals;

/* This example creates a subclass of Node and uses a fancy C++11 lambda
* function to shorten the callback syntax, at the expense of making the
* code somewhat more difficult to understand at first glance. */

class MinimalPublisher : public rclcpp::Node
{
public:
  MinimalPublisher()
  : Node("minimal_publisher"), count_(0)
  {
    publisher_ = this->create_publisher<std_msgs::msg::String>("topic", 10);
    auto timer_callback =
      [this]() -> void {
        auto message = std_msgs::msg::String();
        message.data = "Hello, world! " + std::to_string(this->count_++);
        RCLCPP_INFO(this->get_logger(), "Publishing: '%s'", message.data.c_str());
        this->publisher_->publish(message);
      };
    timer_ = this->create_wall_timer(500ms, timer_callback);
  }

private:
  rclcpp::TimerBase::SharedPtr timer_;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
  size_t count_;
};

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<MinimalPublisher>());
  rclcpp::shutdown();
  return 0;
}
```

Message Publisher

```cpp
#include <memory>

#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

class MinimalSubscriber : public rclcpp::Node
{
public:
  MinimalSubscriber()
  : Node("minimal_subscriber")
  {
    auto topic_callback =
      [this](std_msgs::msg::String::UniquePtr msg) -> void {
        RCLCPP_INFO(this->get_logger(), "I heard: '%s'", msg->data.c_str());
      };
    subscription_ =
      this->create_subscription<std_msgs::msg::String>("topic", 10, topic_callback);
  }

private:
  rclcpp::Subscription<std_msgs::msg::String>::SharedPtr subscription_;
};

int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<MinimalSubscriber>());
  rclcpp::shutdown();
  return 0;
}
```

Message Subscriber

Source

Simple Publisher <=> Subscriber Python

```python
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class MinimalPublisher(Node):

    def __init__(self):
        super().__init__('minimal_publisher')
        self.publisher_ = self.create_publisher(String, 'topic', 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)

    minimal_publisher = MinimalPublisher()

    rclpy.spin(minimal_publisher)

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


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

Message Publisher

```python
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class MinimalSubscriber(Node):

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

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


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

    minimal_subscriber = MinimalSubscriber()

    rclpy.spin(minimal_subscriber)

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


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

Message Subscriber

Source

Simple Publisher <=> Subscriber: Live Example

Let's switch to
the terminal!

Nodes

Definition: *"A node is a participant in the ROS 2 graph, which uses a client library to communicate with other nodes. Nodes can communicate with other nodes within the same process, in a different process, or on a different machine. Nodes are typically the unit of computation in a ROS graph; each node should do one logical thing."*

- Nodes are what you are coding when you are creating a ROS 2 application.
  - Either in C++ or Python using the ROS 2 client libraries *rclcpp* or *rclpy*
- Nodes can publish and subscribe to named topics
- Nodes can act as a service client to have another node perform a computation on their behalf or as a service server to provide functionality to other nodes.
- Nodes can provide configurable parameters to change behavior during run-time.

Parameters

Definition: *“Parameters in ROS 2 are associated with individual nodes. Parameters are used to configure nodes at startup (and during runtime), without changing the code.”*

- The behavior of nodes can be tied to parameters which are loaded during run-time
- Therefore, the node does not need to be re-compiled to cater to different applications.

Topics

Definition: *"Topics are one of the primary styles of interfaces provided by ROS 2. Topics should be used for continuous data streams, like sensor data, robot state, etc. They are anonymous, used in a Publish/Subscribe fashion, and strongly-typed."*

- Publish/Subscribe Architecture
  - producers of data = publishers & consumers of data = subscribers
  - Any published message to topic will be heard by all subscribers
- Anonymous
  - Msg on topic do not (need to) carry information about its sender
- Strongly-Typed
  - A topic can only carry messages of one, a-priori known message type

Messages

Definition: *"Messages are a container in which a ROS 2 node to send data on the network to other ROS nodes, with no response expected."*

- Messages are what is published to topics.
- They are defined in *.msg* files that define each field of a message
- Common message types are built into ROS 2

Services

Definition: *"A service refers to a remote procedure call. In other words, a node can make a remote procedure call to another node which will do a computation and return a result."*

- Like messages passing over a topic without a permanent topic
- Meant for individual instances of information passing:
  - Node 1 (Client) requests information from Node 2 (Server)
  - Node 2 responds with those information
- Expected to return quickly

Actions

Definition: *"An action refers to a long-running remote procedure call with feedback and the ability to cancel or preempt the goal."*

- Like services that are not expected to return quickly
  - Also provided by an action server and used by an action client
- Throughout the execution feedback about the action progress is provided
- Finally returns the result

Additional Resources

- Source for these slides: https://docs.ros.org/en/rolling/Concepts/Basic/
- ROS 2 documentation: https://docs.ros.org/en/rolling/index.html
- Tutorials: https://docs.ros.org/en/rolling/Tutorials.html

- Great YouTube Playlist: https://www.youtube.com/playlist?list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy
- And of course: ChatGPT
  - But: Versions vary! Always make sure the tutorial you are following matches the ROS 2 version you are using.

Conclusion

- ROS 2: The robot operating system
  - Actually a *middleware* that handles takes care of many common tasks in robotics
  - Among other things it provides infrastructure such that
    - *Nodes* that complete tasks can be configured using *parameters*
    - *Nodes* can communicate via *messages* over *topics*
    - *Nodes* can outsource computation via *services* and *actions*
- Installation Guide: https://docs.ros.org/en/humble/Installation.html

- Let‘s crack on with the tutorial!