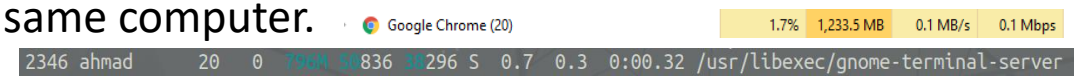
ROS tutorial

Prof. Yu Hue

Ahmad Alghooneh

ROS tutorial - Introduction

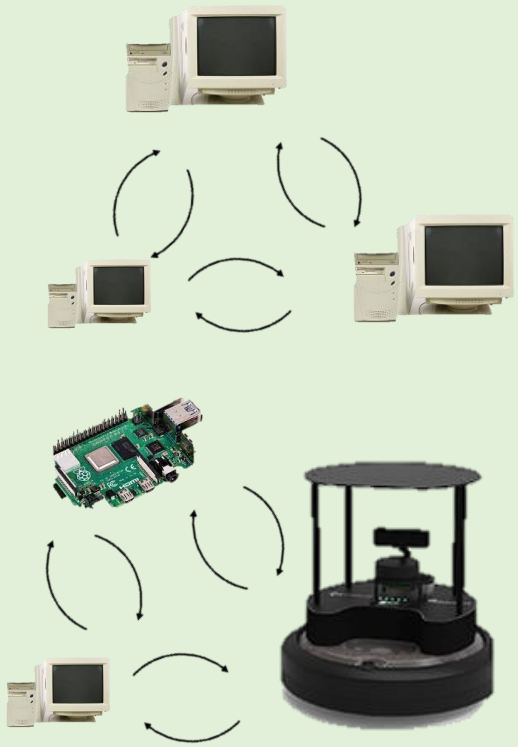
- A Process: In a computer, a process refers to an executing instance of a program or software. It is an independent unit of execution that contains the program code and its current activity. Each process has its memory space, system resources, and execution context, separating it from other processes running on the same computer.



- A Process: Therefore, your code compiled and run is also a process.

- 

A Middleware: is the one who makes the communication between the processes on a local/external machine.



ROS tutorial - Introduction

ROS is a **middleware**, a collection of **libraries/headers/definitions/executables** that help to connect processes both on the local machine or machines across a network.

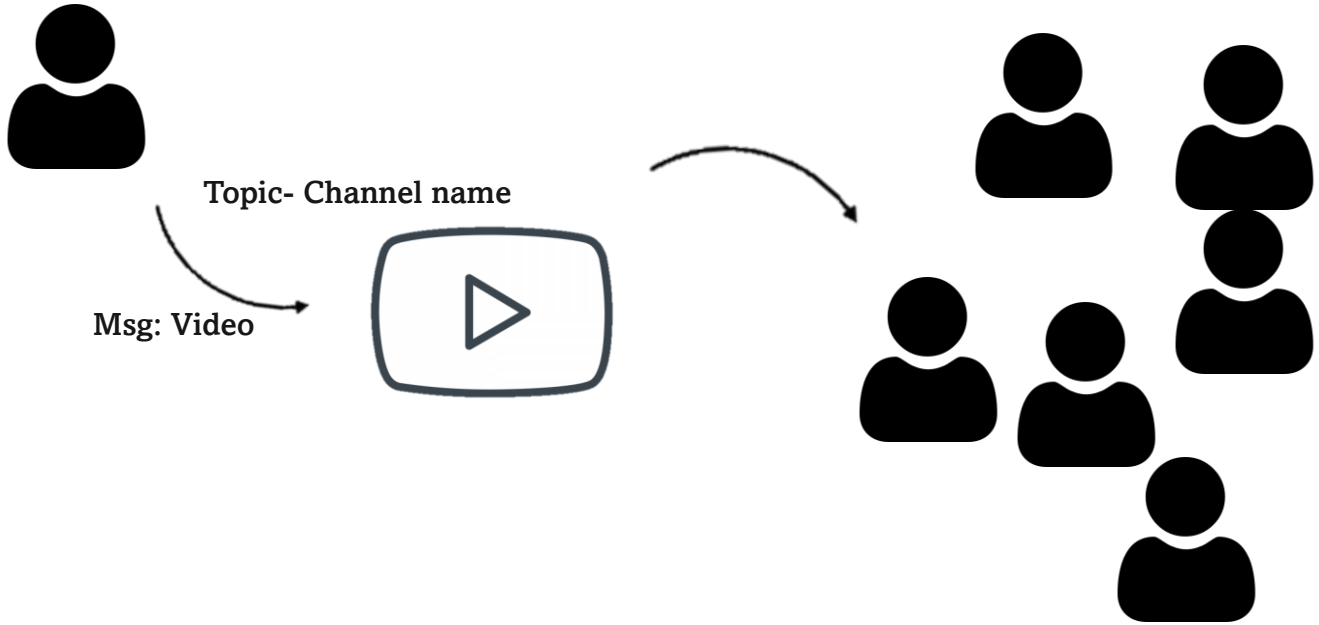
Other than being a middleware, it comes with other **toolboxes and libraries** that make a system able to operate robotics, hence they call it Robotic Operating System

ROS tutorial – Communication

Publishers/Subscriber Scheme

Publishers

Subscribers



Server/Client Scheme

https://www.example.com/page.html?query=example

https://www.youtube.com/results?search_query=zeroMQ

ROS tutorial – Interface, sub/pub

- Topics for ROS, are called rostopics
- It is the medium under which we send information.
- e.g. we can use rostopics to send commands to the robot or receive the position of the wheels.
- After targeting the topic, we should look for the type of message, that we send over the network, we call it rosmsg.
- To see the available topics over the network, we do $>ros2 topic list
- To see the content of a topic, $>ros2 topic echo $topic
- To see the processes that are interfacing with ROS, $>ros2 node list

ROS tutorial – Interface, sub/pub

- To publish a msg on a topic using terminal you can
  $>ros2 topic pub –r $rate $topic $msgType $msg
- If the topic has different fields that we want to see only we should do: $>ros2 topic echo $topic --field field1.field2…fieldn
  e.g. show the position of a odom
  $>ros2 topic echo /odom --field pose.pose
- To check the rate of a topic $>ros2 topic hz $topic
- To see the information about the topic, the msg type, it’s publisher and subscriber: $>ros2 topic info $topic

ROS tutorial – Interface, sub/pub

- To find out about the interfaces used for your ROS installation

  $>ros2 interface list

- You will see three type of interfaces: msgs, services, and actions.
- For sub/pub section we go through msgs. The services and actions are for server/client communication type.
- The msgs are structures defined so it can be parsed on both subscriber and publisher side.
- To show the content of the msg you can do $>ros2 interface show $msg

  ros2 interface show std_msgs/msg/String

ROS tutorial – Interface, sub/pub

- Creating a ROS node is only about writing a script with ros headers included, linked against its libraries.
- The ROS libraries are primarily written in C++, and python, therefore they can be linked against Java, JavaScript, or other languages.
- However, sometimes, we're after creating a package that contains couple of nodes, our own msg/service/action definition, and easy to ship around.
- There we go after ros packages.

ROS tutorial – Script, sub/pub

- In this course, we only write codes with python.
- The most of the ros related classes/interfaces are included in rclpy.
- Now let's write a simple publisher that publishes a msg across ros network.
- This stand alone node can be created with $>vim minPublisher.py
- On top we start by importing the necessary libraries.
- -import rclpy: this will help us to initiate and run the node
- -from rclpy.node import Node this is the type that contains subscription/publisher methods.
- -from std_msgs.msg import String this will help us to send the a standard structure of information, so it's easy to parse on both sides

ROS tutorial – Script, sub/pub

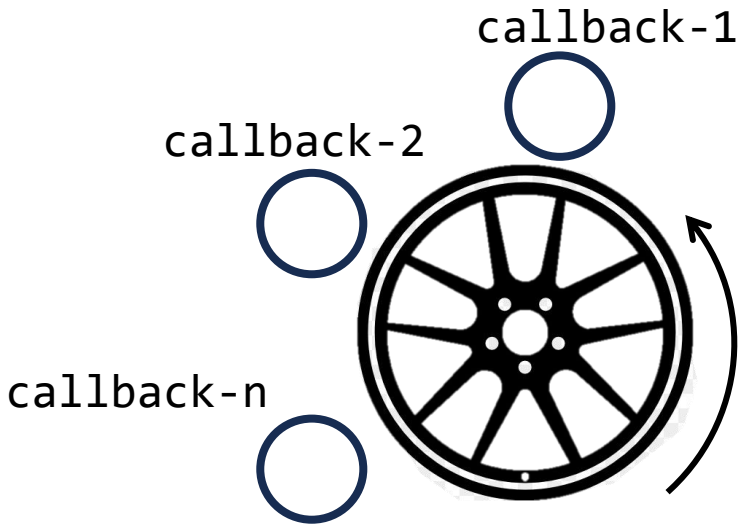
- Now that the necessary libraries are included, we move to the next step, which is creating the publisher.
- It is advised to use classes when writing a Node, this will help us:
  - -Inherit from Node imported, so we can use its methods/properties
  - -We can share variables across the class instance
- Start by –class minimalPublisher(Node)
- - def __init__(self):\ super().__init__("nodeName") this will initialize the parent class.
- -self.publisher_=self.create_publisher(String,'topic',10), this line will write the publisher, that takes the msg type as the first argument, the topic as the second, and QoS as the last. More follows!

ROS tutorial – Script, sub/pub

- To publish the message, we should decide on a rate.
- For this purpose, we define a timer.
- The timer is an event-loop that is triggered at a timer_period
- Just as every other event-loop, it needs a callback.
- The callback will trigger each time the event is happened, which in here is in each timer period.
- -self.timer=self.create_timer(timer_period, self.timerCallback)
- -def timerCallback(self)
- -msg=String(); msg.data='MTE544 is awesome'; self.publisher_.publish(msg)

ROS tutorial – Script, sub/pub

- In the main method, -def main(args=None)
- We initialize the node; this will build the socket to listen on ip/port combination.
- -rclpy.init(args=args); minPublisherInstance=minimalPublisher()
- -rclpy.spin(minPublisherInstance); this will take care of running the ros engine, hence the term spin
- Save the changes by :wq in vim.

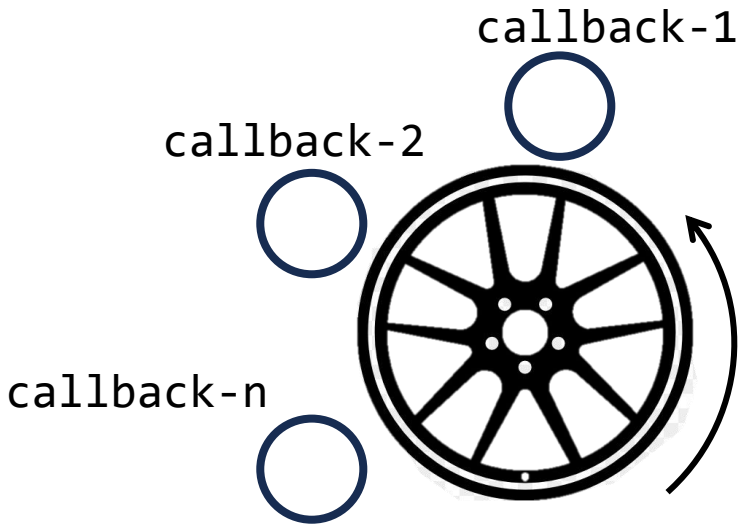


ROS tutorial – Script, sub/pub

- On another terminal, $>vim minSubscriber.py
- Start by –class minimalSubscriber(Node)
- - def __init__(self):\ super().__init__("nodeName") this will initialize the parent class.
- -self.subscription=self.create_subscription(String,'topic',self.subsCallback,10), this line will write the publisher, that takes the msg type as the first argument, the topic as the second, and QoS as the last. More follows!
- -def subsCallback(self, msg);
- -self.get_logger().info('I heard: "%s"' % msg.data)

ROS tutorial – Script, sub/pub

- In the main method, -def main(args=None)
- We initialize the node; this will build the socket to listen on ip/port combination.
- -rclpy.init(args=args); minSubscriberInstance=minimalPublisher()
- -rclpy.spin(minSubscriberInstance); this will take care of running the ros engine, hence the term spin
- Save the changes by :wq in vim.



ROS tutorial – Script, sub/pub

- Open a terminal and then $>python minPublisher.py
- Open another terminal and then $>python minSubscriber.py
- To check the topic is there you can do $>ros2 topic list
- And then, you can see the content of a topic $>ros2 topic echo $topic
- Now you can see that the two nodes are talking to each other on ros network.