



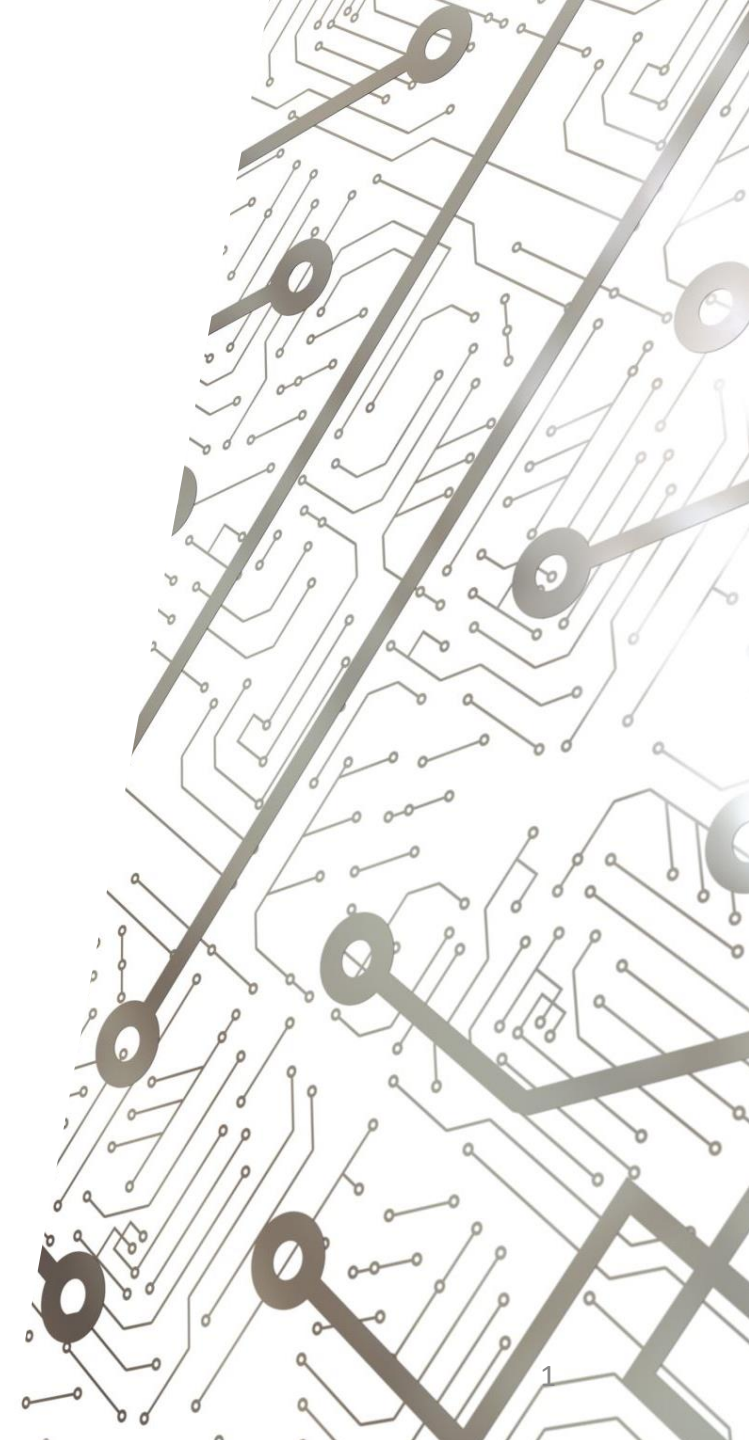
Software Lecture 2:

Getting Started With



ROS

Jacques Cloete



Contents

In this lecture, we shall cover:

- How ROS works (nodes, publishers, subscribers, etc.)
- Installing ROS
- ROSCORE, and running a simple ROS Program
- Creating a workspace to hold our custom packages
- Creating a package to hold our programs and code

Before We Begin

- I strongly suggest bookmarking the following link:
<https://github.com/OxRAMSociety/RobotArm>
- This is the GitHub repository for the robot arm project
- These lectures and all example scripts can be found in:
Tutorials/Software Tutorials (2022)
- Have this accessible while you follow along
- If you download these lecture pdfs, you can copy+paste links and Terminal commands

How ROS Works - Overview

- A ROS network is comprised of many individual programs
- These programs work **together** to achieve a desired function, for example getting a robot arm to move
- To do this, the programs need to be able to **communicate** and send all sorts of data between each other
- This behaviour is characterised by some key terminology...

Nodes, Publishers, Subscribers, Topics

- Each program is run as a **node** in the ROS network
- Nodes communicate by sending **messages** to each other
 - Messages are simple data structures, comprising typed fields (e.g. int32)
- Messages are communicated across **topics**
 - Topics act like communication lines, or data buses, between nodes
- **Publisher** nodes **send out** messages onto a topic
- **Subscriber** nodes **listen to** messages on a topic

Really important to be familiar with this terminology! But don't worry, here's an analogy...

Analogy – Sushi Conveyor Belt

Credit to Sneha Ramshaker!

- Sushi chefs put new plates of sushi on conveyor belts
- Customers at a conveyor belt then take sushi from it



Analogy – Sushi Conveyor Belt

Credit to Sneha Ramshaker!

- Chefs and customers are **nodes**
- Chefs act as **publishers**
- Customers act as **subscribers**
- Conveyor belts are the **topics**
- Sushi itself is the **message**



Some Further Notes

- A node can be both a publisher and/or a subscriber to multiple topics, and you can also easily set up new topics
- **One-way, Many-to-many** communication network
 - When any publisher publishes a message to a topic, **ALL** subscribers to that topic receive the message
ALL customers at the same conveyor belt get a piece of sushi from the new plate!
- Nodes can also interact with each other in more direct ways using ROS actions and services, but we will cover these in later lectures

Time to Install ROS!

<http://wiki.ros.org/noetic/Installation/Ubuntu>

- Load Terminal and follow steps **1.1-1.6** from the linked tutorial to install ROS **Noetic**, noting the following:

1. In step **1.4**, choose the **Desktop-Full** Install

2. In step **1.5**, run the commands listed for **Bash**

*This modifies your **bashrc** file so that the command that sets up ROS-specific Bash commands is automatically called every time you open a new terminal – this will save you from a lot of tedium in the future!*

3. Make sure you complete **1.6**, i.e. install **rosdep**!

sudo apt install python3-rosdep, then run **sudo rosdep init** followed by **rosdep update**

Testing the Installation

- Open a new Terminal and run **roscore**
- You should see something very similar to the following:

```
jacques@JC-Workstation:~$ roscore
... logging to /home/jacques/.ros/log/dbbf55aa-ce4d-11ec-b088-314771286010/ros-l
unch-JC-Workstation-8725.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://JC-Workstation:37707/
ros_comm version 1.15.14

SUMMARY
=====

PARAMETERS
* /rostdistro: noetic
* /rosversion: 1.15.14

NODES

auto-starting new master
process[master]: started with pid [8735]
ROS_MASTER_URI=http://JC-Workstation:11311/

setting /run_id to dbbf55aa-ce4d-11ec-b088-314771286010
process[rosout-1]: started with pid [8745]
started core service [/rosout]
```

What is ROSCORE?

- **ROSCORE** is a collection of core **nodes** and programs that form the foundations of a ROS-based system
- ROSCORE **MUST** be running in order for your ROS nodes to communicate with each other
- Think of it as the underlying web that automatically connects all of your nodes together

Running Your First ROS Program

- Open two more terminals and run the following:

Terminal 2: `roslaunch turtlesim turtlesim_node`

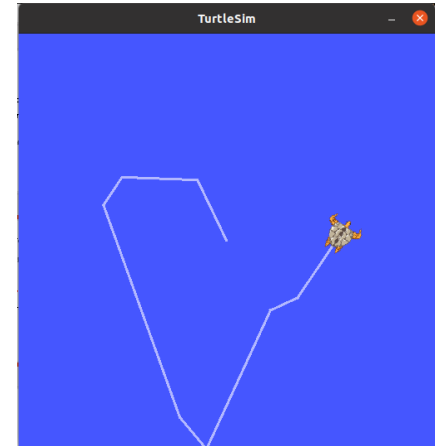
Terminal 3: `roslaunch turtlesim turtle_teleop_key`

- You should see a turtle that can be controlled by your keyboard – try driving it around



Publishers and Subscribers in Action

- The commands you ran set up two **nodes**
- **turtlesim_node** generates the turtlesim environment which acts as both a **subscriber** and a **publisher** – it **listens to** incoming velocity **messages** for the turtle, and **sends out messages** about the turtle's pose (i.e. position and orientation)
- **turtle_teleop_key** acts as a **publisher** – it **sends out** velocity **messages** for the turtle according to your inputs into the keyboard



Publishers and Subscribers in Action

- **Open a fourth terminal and run `rostopic list`**
- This (very helpful) command shows you the names of all **topics** that are currently active
- In this case, the `/turtle1/cmd_vel` topic is used to exchange velocity command messages
- Meanwhile, the `/turtle1/pose` topic is used to exchange messages describing the turtle's pose



Topics and Messages in Action

- We can directly read the contents of messages
- **Now run `rostopic echo /turtle1/pose` and observe the output**
- You will see a stream of messages describing the turtle's live pose, continually being published by the node running the turtlesim
- Click on the Terminal running `turtle_teleop_key` and make the turtle move – see how the messages published change as it moves

```
---  
x: 11.088889122009277  
y: 11.088889122009277  
theta: 0.9808146953582764  
linear_velocity: 0.0  
angular_velocity: 0.0  
---  
x: 11.088889122009277  
y: 11.088889122009277  
theta: 0.9808146953582764  
linear_velocity: 0.0  
angular_velocity: 0.0  
---  
x: 11.088889122009277  
y: 11.088889122009277  
theta: 0.9808146953582764  
linear_velocity: 0.0  
angular_velocity: 0.0  
---  
x: 11.088889122009277  
y: 11.088889122009277  
theta: 0.9808146953582764  
linear_velocity: 0.0  
angular_velocity: 0.0
```

Killing a Program

- To kill a program that is running in Terminal, **press Ctrl+C while that Terminal is selected**
- **Try doing this now for all terminals currently open**
- It is good practice to always do this before closing any Terminal with a program running inside it

Packages and the Workspace

- Programs and software in ROS are neatly organised and bundled together into directories called **packages**
- A (catkin) **workspace** is a folder where you can modify, build, and install these packages
- When you want to use some ROS software, you install the package for it into your workspace

Creating a Workspace

Exit all Terminals and load up a new one;

- Install catkin: **sudo apt install ros-noetic-catkin python3-catkin-tools python3-osrf-pycommon** (all one line!)
- Also install wstool: **sudo apt install python3-wstool**
- Now create a directory to host the workspace:
mkdir -p ~/tutorial_ws/src
- Navigate to the source folder: **cd ~/tutorial_ws/src**

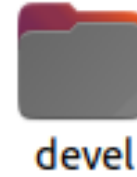
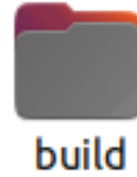
Creating a Workspace

- Run the following command to add missing dependencies:
rosdep install -y --from-paths . --ignore-src --rosdistro noetic
- Navigate to the main workspace directory: **cd ~/tutorial_ws**
- Run the following: **source /opt/ros/noetic/setup.bash**
- Configure the properties for the workspace:
catkin config --extend /opt/ros/noetic
- Build the workspace: **catkin build**

Sourcing the Workspace

- We must also source the workspace; to do this, run this line:
`source ~/tutorial_ws/devel/setup.bash`
- Add the above command to your bashrc to automate it:
`echo 'source ~/tutorial_ws/devel/setup.bash' >> ~/.bashrc`

The Workspace



- The workspace is made up of four main directories
- **Source space** contains the source code for all your installed and custom packages – this is where we will create new packages
- The other three directories are important for the functionality of the workspace but not covered here

An Aside – Building the Workspace

- Why did we have to build the workspace?
- **Building** is the process of compiling source code into stuff like executable programs
- In our case, each ROS package needs to be built before we can use the stuff they provide
- The **catkin build** command automatically builds every package in the workspace whilst handling the interactions and dependencies between packages when building!

Creating a Package

- To properly start making use of ROS, we will write our own programs and code – we will do this in custom packages
- **Navigate into the Source folder:** `cd ~/tutorial_ws/src`
- **Create a new package:**
`catkin_create_pkg tutorial_scripts std_msgs rospy`
- The first argument is the package name, and all following arguments are the dependencies, i.e. the other packages required for use by the new package

Creating a Package

- Now return to the main workspace directory (**cd ..**) and **rebuild** the workspace: **catkin build**
- Now we have a custom package ready to house our own programs and code!
- Note that every time you add a new package to the workspace, you will need to rebuild the workspace to incorporate that new package

Downloading & Installing Packages

- There are many packages freely available for you to install, providing all sorts of useful functionality
- Before installing a package, always update your Debian package index: **sudo apt update**
- To find available packages: **apt search ros-noetic**
- To install a package: **sudo apt install name-of-package**

Final Note: The ROS Wiki

- <http://wiki.ros.org/Documentation>
- **Incredibly** useful source of knowledge for all things ROS
 - Plenty of documentation on core ROS functionality
 - Info on many useful packages
 - Tutorials and example code to help you learn

Summary

We covered:

- How ROS works (nodes, publishers, subscribers, etc.)
- Installing ROS
- ROSCORE, and running a simple ROS Program
- Creating a workspace to hold our custom packages
- Creating a package to hold our programs and code

Homework: Familiarise yourself with ROS terminology (p.5), finish setting up workspace

Next time, we will code up our first ROS program!

Thank You!

Any Questions? Contact jacques.cloete@trinity.ox.ac.uk