مادة نظم التشغيل
محاضرة /٢/ عملي

م. باهر خير بك

Lab Session 2
ROS Workspace and ROS Packages

1. ROS Workspace

A ROS workspace is a directory that contains one or more ROS packages. A workspace is like a project folder where you can organize your ROS packages and build them together.
To create a ROS Workspace, follow the steps:

1. Open a new terminal window

2. Create a new directory for your workspace (typically named catkin_ws but you can choose any name you wish) and append another directory within called src

3. Navigate to your new directory and run the command $catkin_make

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws
$ catkin_make
```

After the workspace builds, two new folders will be created inside the /catkin_ws directory. One is the /devel directory which is where ROS dumps all the built files. The second is the /build which is where we run $cmake from to build the packages in the /src folder. The /src directory that we created will house all of the ROS Nodes (We will go in depth about these topics in the next session)

Inside the /devel folder there is a setup.bash executable file that we have to manually source in each terminal session in order to overlay our workspace on top of our ROS environment.

```
$ source devel/setup.bash
```

The following tree demonstrates the file structure of a ROS Workspace

```
catkin_ws/               -- WORKSPACE
  src/                   -- SOURCE SPACE
    ...
  build/                 -- BUILD SPACE
  devel/                 -- DEVEL SPACE
    setup.bash          \
    setup.sh            |-- Environment setup files
    setup.zsh           /
    etc/                 -- Generated configuration files
    include/             -- Generated header files
    lib/                 -- Generated libraries and other artifacts
      package_1/
        bin/
        etc/
        include/
        lib/
        share/
        ...
      package_n/
        bin/
        etc/
        include/
        lib/
        share/
    share/               -- Generated architecture independent artifacts
    ...
```

2. ROS Package

The ROS packages are the most basic unit of the ROS software. They contain the ROS runtime process (nodes), libraries, configuration files, and so on, which are organized together as a single unit. You can think of a ROS Package as one standalone project.
To create a ROS Package, follow the steps:

1. Open a new terminal window
2. $cd into the src folder inside the workspace that we created.
3. Use $catkin_create_pkg command to create a new package and add dependencies
4. Build the package in the catkin workspace using $catkin_make

```
$  cd ~/catkin_ws/src
$  catkin_create_pkg lecture2 std_msgs rospy roscpp
$  cd ..
$  catkin_make
```

`$catkin_create_pkg` requires that you give it a `package_name` and optionally a list of dependencies on which that package depends

```
# This is an example, do not try to run this
# catkin_create_pkg <package_name> [depend1] [depend2] [depend3] .....
```

The dependencies are all the elements that our package will depend on. In our case, we will want to write our nodes in both C++ and Python which is why we added the roscpp and rospy dependencies. The std_msgs dependency contains common message types representing primitive data types and other basic message constructs, such as Int, String and multiarray.

The following tree demonstrates the file structure of a ROS Package

```
workspace_folder/         -- WORKSPACE
  src/                    -- SOURCE SPACE
    CMakeLists.txt        -- 'Toplevel' CMake file, provided by catkin
    package_1/
      CMakeLists.txt      -- CMakeLists.txt file for package_1
      package.xml         -- Package manifest for package_1
    ...
    package_n/
      CMakeLists.txt      -- CMakeLists.txt file for package_n
      package.xml         -- Package manifest for package_n
```

inside the package, there are 2 new files that were created each with distinct purposes. The package.xml file is an essential file used in the Robot Operating System that defines the properties and metadata of a ROS package. This file defines properties about the package such as the package name, version numbers, authors, maintainers, and dependencies on other catkin packages.

The following elements are what we need to pay attention to:

- DESCRIPTION TAG

```
<description>The lecture2 package</description>
```

Change the description to anything you like, but by convention the first sentence should be short while covering the scope of the package. If it is hard to describe the package in a single sentence then it might need to be broken up.

- **MAINTAINER TAGS**

```
<!-- One maintainer tag required, multiple allowed, one person per tag -->
 <!-- Example:  -->
 <!-- <maintainer email="baher.kherbek@outlook.com">Jane Doe</maintainer> —>
  <maintainer email="user@todo.todo">user</maintainer>
```

This is a required and important tag for the package.xml because it lets others know who to contact about the package. At least one maintainer is required, but you can have many if you like. The name of the maintainer goes into the body of the tag, but there is also an email attribute that should be filled out.

- **DEPENDENCIES TAGS**

```
  <buildtool_depend>catkin</buildtool_depend>
   <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>std_msgs</build_depend>
 <exec_depend>roscpp</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>std_msgs</exec_depend>
```

<build_depend> and <exec_depend> tags are generated based on the dependencies that we generated in our package. We will also be editing these dependencies tags in later sessions.

Next is the CMakeLists.txt which contains a set of directives and instructions describing the project's source files and targets (executable, library, or both) which represents the input to the CMake build system for building software packages. In future sessions, we will constantly be editing our CMakeLists.txt file to match our package needs such as custom messages and services.

3. Bashrc file and ROS commands

The .bashrc file contains a set of commands that get executed upon the launch of a new terminal session. We can setup our .bashrc file to automatically $source our ROS Workspace without having to source it manually in each individual session.

There are two options, either open the .bashrc file using gedit and add the $source command manually or we can append it directly using the $echo command like so:

```
$  echo ‘source ~/catkin_ws/devel/setup.bash’ >> ~/.bashrc
$  source ~/.bashrc
```

Now we can neglect having to $source our workspace each time because it will have been sourced automatically upon the launch of a terminal window. After the workspace has been sourced we can run ros related commands on our packages. $roscd and $rospack are two main ones. The $roscd command allows us to immediately navigate to our package directory without having to manually $cd into the entire path. So instead of typing out:

```
$  cd ~/catkin_ws/src/lecture2
```

We can call the $roscd command anywhere no matter where our current directory is in our terminal

```
$  roscd lecture2
```

But make sure the workspace is sourced otherwise the package will not be found.

The $rospack commands displays all the dependencies of a specific package

```
$  rospack depends1 lecture2
```