# ROVI2 - Exercise for ROS Lecture

Thorbjørn Mosekjær Iversen (thmi@mmmi.sdu.dk) and
Stefan-Daniel Suvei (stdasu@mmmi.sdu.dk)

February 2, 2017

## 1 Introduction to exercise

The purpose of these exercises is getting you started using the ROS framework and to use it in combination with RobWork, OpenCV and PCL. The exercises are structured as follows. The first section, section 2, is an overview of a bunch of ROS tutorials that you should go through to get a basic understanding of the framework. In section 3 you should try out the example code we provide and inspect the code to get an understanding of the communication between nodes. In section 4 you should create your own ROS package and make nodes that communicate with the robot platform in RoboLab. Finally in section 4, some notes on creating your own package in the virtual machine for communication with the RoboLab platforms. It is suggested to do as follows:

1. Go through the ROS tutorials
2. Inspect and run the provided code
3. Create a new package to communicate with the robot platform in RoboLab and create nodes to receive images and pointclouds as well as a node to communicate with the robot.
4. Try out your node at the robot platform and verify that you can control the robot and receive sensor data.

The steps are explained in more details in the sections below.

**Note on virtual machine** The exercise is made for the virtual machine and it is suggested to use this as it will make life easier for you. Secondly it should be noted as already pointed out earlier, that we do not provide support for other platforms than the virtual machine. A folder denoted catkin_ws is already present in the virtual machine home directory. It is suggested you delete this such that you start out with clean catkin workspace that you make by yourself.

## 2 Suggested ROS tutorials

It is suggested that at a minimum you do at least the tutorials listed below. For some of the tutorials, a note is added about what to focus on.

The ROS tutorials can be found at `www.ros.org` ⇒ wiki ⇒ Tutorials or go to: `http://wiki.ros.org/ROS/Tutorials`

The ROS tutorials are quite comprehensive and tend to go into details with some of the less important features and commands as well as the important ones. A list of suggested tutorials and what to focus on in the tutorials is therefore given below.

### Beginner Level

**1. Installing and Configuring Your ROS Environment**

**3. Creating a ROS package** This tutorial may be skipped in the beginning, but it is worth looking here, when you have to create your own package.

**4. Building a ROS Package** The tutorial in itself is less important, but the important thing to notice in this tutorial is that you don't build ROS using the traditional make commands. But instead using the *catkin_make* command. This command is executed within the root of the catkin workspace.

**5. Understanding ROS Nodes** Focus on roscore and rosrun commands
**roscore** : starts the ROS kernel
**rosrun [package_name] [node_name]** : starts ROS node [node_name] from ROS package [package_name]

**6. Understanding ROS Topics** This tutorial covers a lot of ways to extract data about the communication between ROS nodes. Its enough to read through this tutorial but take special care and tryout the commands explained in section: 1.2.2 , 1.2.3 , 1.2.4 , 1.3 as these are used fairly often when debugging and using ROS.

**7. Understanding ROS Services and Parameters** Sort of the same as above. Just read through this tutorial and notice that **rosservice list** gives you a list of the services running.

**10. Creating a ROS msg and srv**

**11. Writing a Simple Publisher and Subscriber (C++)**

**14. Writing a Simple Service and Client (C++)**

**17. Recording and playing back data** This is not important now, but its a nice feature, which can come in handy at a later point in your projects. Just notice what this feature is all about so that you know it exist.

### Intermediate level

**4. Running ROS across multiple machines** This is not important now, but you will need this at a later point when using ROS across multiple computers (at the robot platforms in RoboLab). Notice that this tutorial explains how to do this.

## 3 Inspecting and running simulated sensor code

A code example is provided where ROS, RobWork and OpenCV are all used in the same project. The code can be obtained from Blackboard. It already compiled in the catkin workspace on the virtual machine.

The example consist of two things:

1. A set of ROS packages for simulating a camera
2. Workcell descriptions for RobWork

## 3.1 ROS packages

The provided packages are listed below.

**monocamera_dummy** A package with a node from where images can be extracted through ROS. The node is supposed to simulate a real camera and has the same interface as will be used on the real workcell.

**camera_plugin** Contains a plug-in for RobWorkStudio. The plug-in is able to show the camera images published by monocamera_dummy

## 3.2 The workcell description:

Models of the physical workcells in RoboLab. For this exercise you should use the UR5 workcell as the plugins are made for this. When you continue creating your own package for the communication with the real robot you should utilize the workcell file according to the workcell you are working on.

## 3.3 Using and compiling the source code

The source code and workcells are available through Blackboard. Start by creating and initialising the catkin workspace, on the virtual machine, using **catkin_init_workspace** in the **catkin_ws/src** folder. If you have already done that as a part of the tutorials you can skip this step and directly copy the packages from the source code files into your catkin workspace src-folder. Next, we need to ensure that ROS is sourced properly. Do this by editing your .bashrc file in your home directory. In a new terminal (ctrl+alt+t), type:

```
gedit .bashrc
```

Add the following line or lines to the end of the file if they are not already there:

```
# ROS
source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash
```

By doing so these are sourced when you start at new terminal. To verify that the ROS_PACKAGE_PATH is setup correctly, open a new terminal and type:

```
echo  $ROS_PACKAGE_PATH
```

This should result in something similar to this:

```
/home/student/catkin_workspace/src:/opt/ros/kinetic/share:/opt/ros/kinectic/stacks
```

When the paths for ROS is setup properly. Then go to the root of the workspace and build the workspace using **catkin_make**. If the compilation is terminated due to inability to find RobWork, RobWorkStudio and RobWorkHardware you should add the following lines to your .bashrc file:

```
export RW_ROOT=~/RobWork/RobWork
export RWHW_ROOT=~/RobWork/RobWorkHardware
export RWS_ROOT=~/RobWork/RobWorkStudio
```

Remember to restart the terminal such that the .bashrc file is sourced and retry compilation.

After the workspace is successfully compiled go to the binary folder of RobWorkStudio (RobWork/RobWorkStudio/bin/release). Modify the RobWorkStudio.ini configuration file such that the camera_plugin is loaded on start. The configuration files should point to the plugin, which is located in **catkin_ws/devel/lib**. Alternatively you can load the plugin manually from RobWorkStudio by choosing menupoint: plugins ⇒ load plugin and then choose the 'camera plugin' which will have appeared in the plugin drop down menu.

### 3.4 Launching the code

In order to launch the system do as follows:

```
# Run the roscore
roscore

# Open another terminal and run monocamera_dummy
rosrun monocamera_dummy monocamera_dummy

#Optionally: open new terminal and check that images are being published
rosrun rqt_image_view rqt_image_view

# Open another terminal and go to RobWorkStudio binary directory
cd ~/RobWork/RobWorkStudio/bin/release

# Run RobWorkStudio with parameters if you want to load a workcell and
# a specific RobWorkStudio.ini file.
./RobWorkStudio --ini-file RobWorkStudio.ini "<path-to-workcell-file>"

# Otherwise run without parameters then it will use the RobWorkStudio.ini
# file located  from where you call RobWorkStudio
./RobWorkStudio
```

Play around with the program and plugins. Start a new terminal where you use some of the ROS commands to observe the communication, such as **rosservice lists** and **rostopic lists**. You should also try to look at some of the ROS graphical tools such as **rqt** and **rqt_graph**, **rviz** etc.

Finally you should have a look at the source code and examine how messages are defined and how the ROS communication is setup.

## 4 Creating a new package and connecting to the robot setups in RoboLab

The last exercise is where you are to make use of the knowledge you have acquired in the previous ROS-tutorials and the example plugin for the virtual machine and RobWork. The aim of this exercise is to give you a good starting point for the future project work on the real platforms. You should follow the steps as they come:

## Connect to robot platform

1. Connect to the robots and sensors in Robolab. Please refer to the guide: *Connect_to_hardware.pdf*

## Getting started

1. Create a ROS package

## Communicate with the cameras

2. Write a ROS node that capture images from the BumbleBee2 stereo camera and saves the images to your computer. Alternatively you can capture the images published by the monocamera_dummy node. The published message is of the type **sensor_msgs::Image**

3. Write a ROS node (Could be the same as the one capturing stereo images) that capture PointClouds from the RGB-D device and saves the pointclouds to your computer. For additional information of dealing with PointClouds in ROS, see `http://wiki.ros.org/pcl/Overview`. The messages that are outputted from the robot platform is of the type **sensor_msgs::PointCloud2**. Be aware that if you want to use 'Point Cloud Library' in ROS your node should depend on the package **pcl_ros**.

## Communicate with the robots

Before doing this exercise, make sure you have installed the latest version of RobWork, RobWorkStudio and RobWorkHardware. A guide on how to install these can be found on Black Board.

The next step is to download and build the CAROS package in your catkin workspace. This can be done by following these steps:

```
cd ~/catkin_ws/src
git clone https://gitlab.com/caro-sdu/caros.git
cd ~/catkin_ws
catkin_make
```

Once CAROS is built, you need to replace a header file inside the *caros_universalrobot* package, in order for the collision checker to function correctly. The new header file can be found on BlackBoard. Download it and then replace the old one from:
`~/catkin_ws/src/caros/hwcomponents/caros_universalrobot/include/caros/test`

To download the UR5 controller simulator, go to
`http://www.universal-robots.com/download/?option=16594#section16593`, select the desired software version of the controller in the `Select Software version`, click `Download` and then follow the install instructions:

- Create a new directory in your workspace
- Go to folder:`cd --name_of_folder--`
- Uncompress the software: `tar xvzf --name_of_file--`
- Go to software folder:`cd ursim-3.X`

- Run the install script:`./install.sh`
- Run software:`./ursim-3.X/start-ursim.sh`

X represents the software version number. NOTE: You might run into running problems with some of them. If it fails, try a different version.

Once the controller simulator starts, click `OK` on `Go to initialization screen`, than go to `Run Program -> Move` and then click on `Home` to move the robot joints into their Home position.

To work with to the robot, follow these steps:

- Change the IP address of the robot and of the PC in the parameters .xml file in `catkin_ws\src\caros\hwcomponents\caros_universalrobot\launch`

  The *device_ip* is the robot's IP address and the *callback_ip* is your laptop's own ip address. When running with the simulator, *device_ip* and the *callback_ip* should be the same. You can find the correct address by opening a terminal and running `ifconfig` (check the *inet addr* from *lo*).

- Open a terminal and run the *caros_universalrobot* launch file with:
  `roslaunch caros_universalrobot simple_demo_using_move_ptp.test`

You will see the robot move to a specific waypoint and backwards in the controller simulator. If all the above steps are performed successfully, you can now try to:

4. Write a ROS node that reads out the configuration of the robot. You will find the proper service file(s) on Blackboard.
5. Choose three points that the Robot should move between. Be aware that the robot can and will collide with the environment if your choices are bad. Make the robot move between the three points.
6. While moving the robot between the three points, record the configuration during the path and plot it to see the actual path compared to the three points you put into it.

## 5 Acknowledgement

Based on tutorial by Johan Sund Larsen (josl@mmmi.sdu.dk), Thomas Fridolin Iversen (thfi@mmmi.sdu.dk) and Mikkel Tang Thomsen (mtt@mmmi.sdu.dk)