Introduction to Unmanned Ground Vehicles

Instructor: Micho Radovnikovich
Email: mtradovn@oakland.edu
ECE 495/595 – Winter 2017

Homework #2 – ROS Fundamentals

Due: Wednesday, January 25th

The purpose of this assignment is to introduce writing simple ROS nodes. Additionally, the procedure of running a provided unit test to validate the implementation will be introduced.

Provided Files

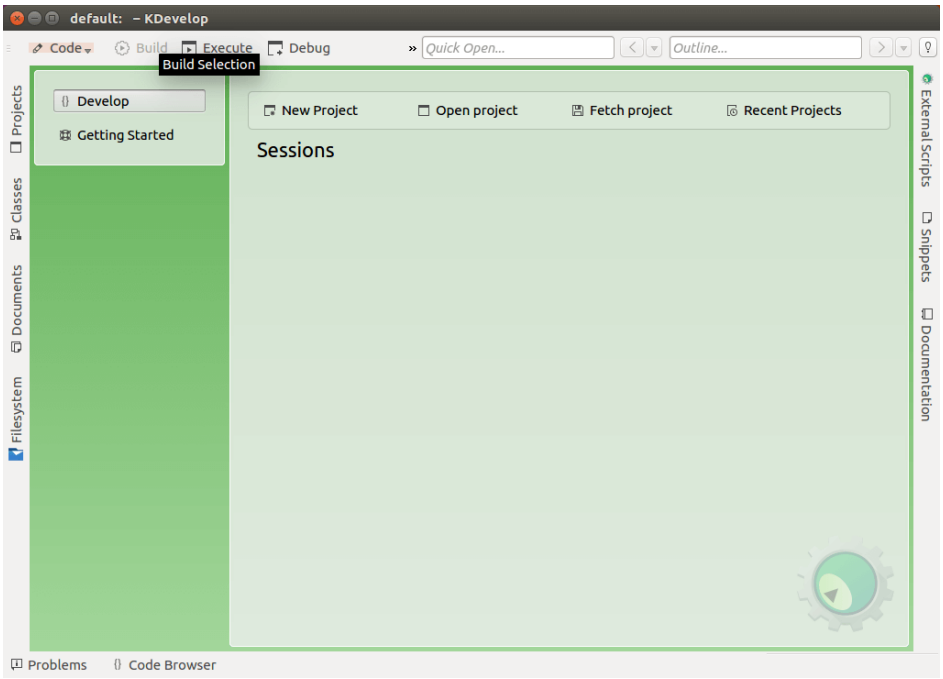
The following resources will be provided in the central repository when Homework #2 is assigned:

- Pre-configured **package.xml** and **CMakeLists.txt** files for the **homework2** package.
- Template C++ files for the nodes.
- Test script to validate the nodes.
- Launch file to run test scripts.

Step 0 – Set Up KDevelop

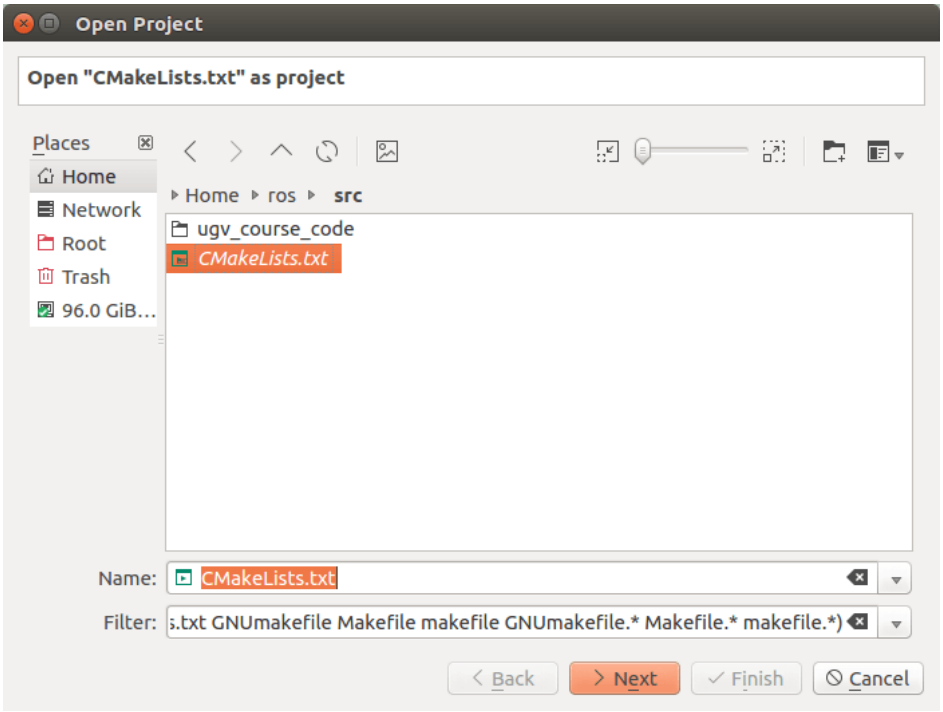
KDevelop is the Integrated Development Environment (IDE) that we'll use in the course. There are many other IDEs available, and if you're interested, you can check out the other ones that people have used with ROS: `http://wiki.ros.org/IDEs`.

1. Open KDevelop using the provided Desktop launcher. **Don't open KDevelop from the program menu!**

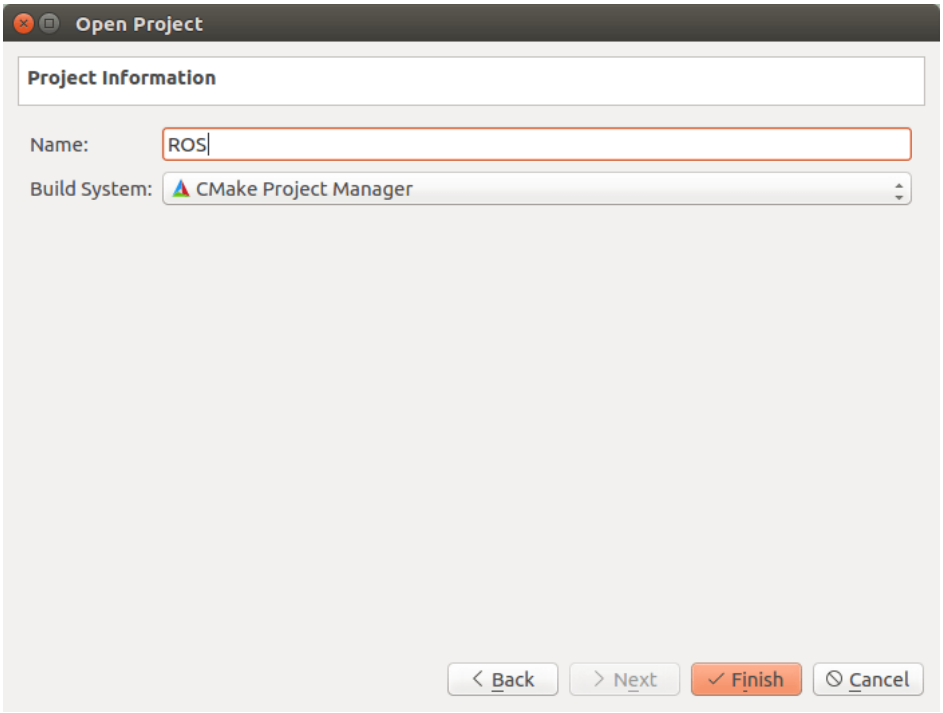


2. In the **Project** menu, click **Open / Import Project**.

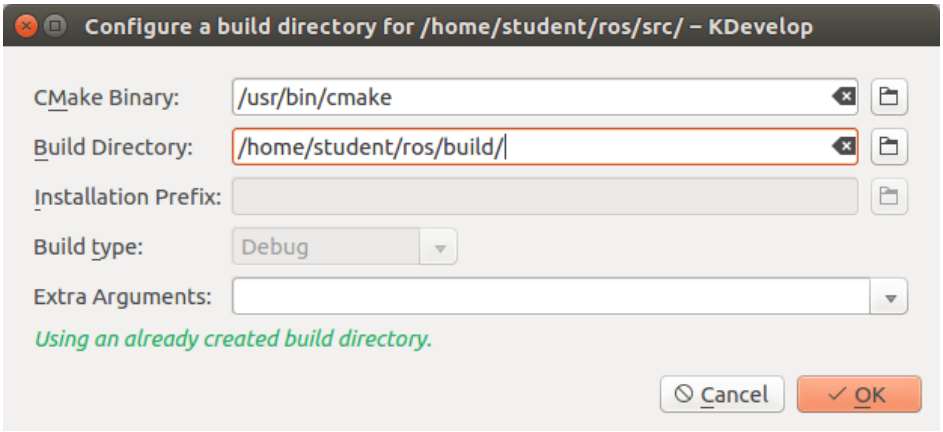
3. Navigate to the **src** folder in the ROS workspace folder, select **CMakeLists.txt** and click **Next**.



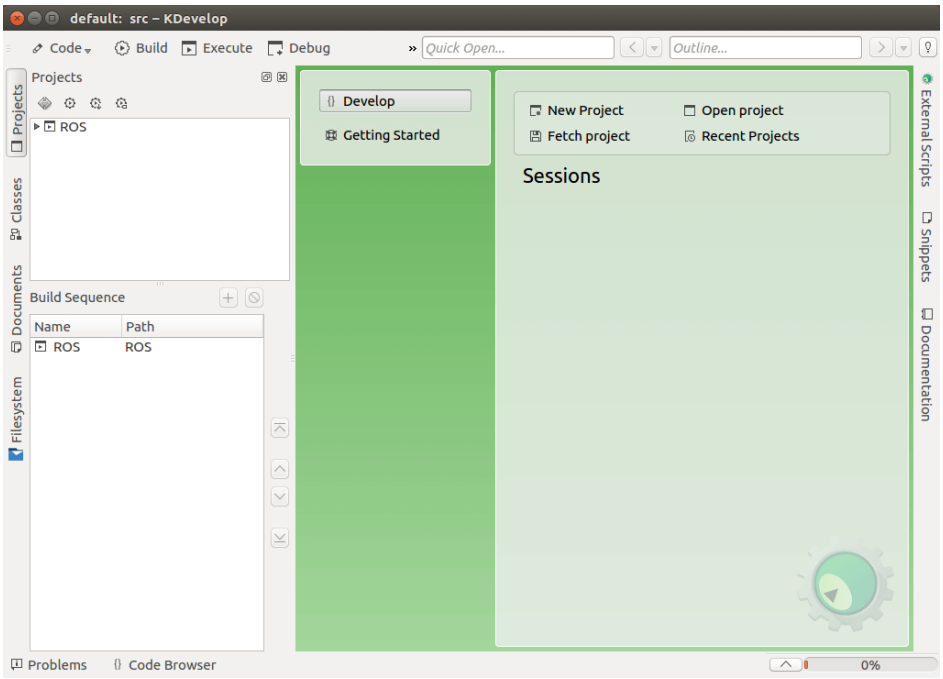
4. On the next screen, put **ROS** in the **Name** text box, and click **Finish**.



5. Make sure the next screen looks like the following. If you're not using the provided VM, **student** will be replaced by your Ubuntu account name. Finally, click **OK**.



6. The KDevelop window should look like the following. If you don't see the project list pane, click **Projects** on the top of the left side of the screen.



Step 1 – Write the Publisher/Subscriber Node

Referring to examples in the course notes and the ROS tutorial on the subject (http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29), modify the template file **pubsub_node.cpp** to do the following:

> Subscribe to a **std_msgs/Float64** topic called **/topic_in**.

> In the topic receive callback, add 5 to whatever value was received and publish the result on another **std_msgs/Float64** topic called **/topic_out**.

Step 2 – Create the Service

To create a service, you have to create the **srv** file, define the request and response structures, and configure **CMakeLists** to build the headers for the service. A good tutorial on how to do this can be found on the ROS wiki (http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv#Creating_a_srv). Referring to the course notes and this tutorial, do the following:

> Create a service file called **StringCat.srv** in a new **srv** folder in the root of the **homework2** package.

> This service should have a **string** value called **string_in** as a request, and another **string** value called **string_out** as a response.

> Add the appropriate text to the **CMakeLists.txt** file of the **homework2** package to add the service file and to generate messages.

Step 3 – Write the Service Advertiser

Next, write a ROS node to advertise a **StringCat** service. You can find another tutorial on the ROS wiki to do this (http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29). Referring to the course notes and this tutorial, modify the template file **service_advertiser_node.cpp** to do the following:

> Advertise a **StringCat** service called **/concatenate_string**.

> In the service callback, take **string_in** from the request and concatenate the string " **with more text**" to the end (mind the leading space).

> Put the resulting string into **string_out** in the response.

Step 4 – Write the Service Client

Finally, write another ROS node to call the service advertised by the node written in Step 3. Referring to the course notes and the ROS wiki tutorial, modify the template file **service_client_node.cpp** to do the following:

> Create a **serviceClient** that points to the **/concatenate_string** service.

> Populate a request structure with an arbitrary string in the **string_in** field.

> Call the service and report the result to the terminal with a **ROS_INFO**.

> Terminate the program by letting it return from **main**.

Step 5 – Test the Nodes

First, be sure to compile the ROS workspace with **catkin_make** before trying to run anything! Then, test your implementation by running the provided test launch file:

```
roslaunch homework2 test_homework2.launch
```

This will test the publisher / subscriber node and the service advertiser node for proper operation. If everything works correctly, the tests should pass, and the terminal window should look something like this:

Next, test your service client node. Before running the client, be sure the service advertiser node is running. When you run the service client node, the service call should succeed, and the result shown in the terminal should contain the appropriately concatenated string.

What to Submit

To submit your homework, do the following:

> Verify that the automated tests finished without any failures.

> Verify that the service caller node dumps the correct string to the terminal.

> Once everything is running well, be sure to commit all your changes to your fork of the central repository:

- Modified versions of **pubsub_node.cpp**, **service_advertiser_node.cpp** and **service_client_node.cpp**
- Modified version of **CMakeLists.txt**
- Added **StringCat.srv**

> Add a commit message describing the fact that you are submitting Homework #2, and push the revision to BitBucket.