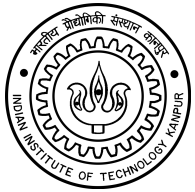


ROS

ROBOTICS CLUB

SCIENCE AND TECHNOLOGY COUNCIL
IIT KANPUR

Robotics
Club IITKanpur

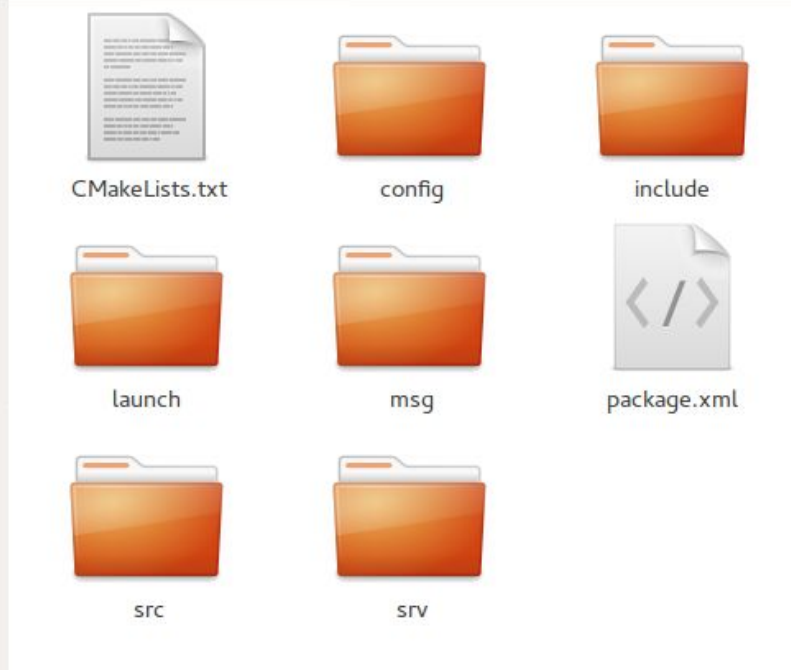
Contents

- Understanding the *ROS Package Structure*
- Creating *Packages*
- Writing/Editing *package.xml*
- Understanding *CMakeLists.txt*
- Writing *Publisher* and *Subscriber Nodes*
- Understanding the *parameter server*
- Writing *.param files*

Robotics Club IITKanpur

ROS Packages

- ROS software is organized into *packages*, which can contain source code, *launch files*, configuration files, *message definitions*, data, and documentation
- A package that builds up on/requires other packages (e.g. message definitions), declares these as dependencies



Robotics Club IITKanpur

ROS Packages - the Directories

The `src` directory:
Contains the definitions of all of your *nodes* (source files).

The `include/package_name` directory:
Contains the C++ header (#include) files.

The `config` directory:
Contains all the *parameters* for your package (written in YAML).

The `launch` directory:
Contains all your *launch* files.

The `msg` directory:
Contains your *custom message* definitions.

The `srv` directory:
Contains your *custom service* definitions.

Robotics Club IITKanpur

package.xml

The `package.xml` file defines the properties of the package, like:

- Package name
- Version number
- Authors
- **Dependencies on other packages**

package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>ros_package_template</name>
  <version>0.1.0</version>
  <description>A template for ROS packages.</description>
  <maintainer email="pfankhauser@any…">Peter Fankhauser</maintainer>
  <license>BSD</license>
  <url type="website">https://github.com/leggedrobotics/ros_…</url>
  <author email="pfankhauser@anybotics.com">Peter Fankhauser</author>

  <buildtool_depend>catkin</buildtool_depend>

  <depend>roscpp</depend>
  <depend>sensor_msgs</depend>
</package>
```

Robotics Club IITKanpur

Editing *package.xml*

These six types of dependencies are specified using the following respective tags:

- `<build_depend>`
- `<build_export_depend>`
- `<exec_depend>`
- `<test_depend>`
- `<buildtool_depend>`
- `<doc_depend>`

`<depend>` specifies that a dependency is a *build*, *export*, and *execution* dependency. This is the most commonly used dependency tag.

You will have to use these tags in the `package.xml` file to add dependencies.

Robotics Club IITKanpur

CMake

- Remember, catkin runs on **g++, python** and **CMake**
- CMake is a **buildtool** - build systems use it to do the building
- **Compiler independent** - will be the same for Python or C++
- Basically gives ‘high-level’ instructions - where to find libraries, which compiler to use, etc.
  - **g++** and **Python** do the actual building
- Can say it is a **language** in itself
- Needs a **CMakeLists.txt** file to run

Should be pre-installed, but still: **sudo apt-get install cmake**

CMakeLists.txt

Major components:

1. Required CMake Version (`cmake_minimum_required`)
2. Package Name (`project()`)
3. Find other CMake/Catkin packages needed for build (`find_package()`)
4. Message/Service/Action Generators (`add_message_files()`, `add_service_files()`, `add_action_files()`)
5. Invoke message/service/action generation (`generate_messages()`)
6. Specify package build info export (`catkin_package()`)
7. Libraries/Executables to build (`add_library()`/`add_executable()`/ `target_link_libraries()`)
8. Tests to build (`catkin_add_gtest()`)
9. Install rules (`install()`)

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(ros_package_template)

## Use C++11
add_definitions(--std=c++11)

## Find catkin macros and libraries
find_package(catkin REQUIRED
  COMPONENTS
    roscpp
    sensor_msgs
)

...
```

Robotics Club IITKanpur

Editing *CMakeLists.txt*

To add package dependencies and executables, you have to edit the following functions:

- `find_package()`
- `add_executable()`
- `target_link_libraries()`

CMakeLists.txt Example

```
cmake_minimum_required(VERSION 2.8.3)
project(husky_highlevel_controller)
add_definitions(--std=c++11)

find_package(catkin REQUIRED
  COMPONENTS roscpp sensor_msgs
)

catkin_package(
  INCLUDE_DIRS include
  # LIBRARIES
  CATKIN_DEPENDS roscpp sensor_msgs
  # DEPENDS
)

include_directories(include ${catkin_INCLUDE_DIRS})

add_executable(${PROJECT_NAME} src/${PROJECT_NAME}_node.cpp
src/HuskyHighlevelController.cpp)

target_link_libraries(${PROJECT_NAME} ${catkin_LIBRARIES})
```

Use the same name as in the `package.xml`

We use C++11 by default

List the packages that your package requires to build (have to be listed in `package.xml`)

Specify build export information

- `INCLUDE_DIRS`: Directories with header files
- `LIBRARIES`: Libraries created in this project
- `CATKIN_DEPENDS`: Packages dependent projects also need
- `DEPENDS`: System dependencies dependent projects also need (have to be listed in `package.xml`)

Specify locations of header files

Declare a C++ executable

Specify libraries to link the executable against

Robotics Club IITKanpur

Creating a ROS Package

Inside the **src** folder of your workspace,

```
catkin_create_pkg [PACKAGE_NAME] [DEPENDENCIES]
```

- [DEPENDENCIES]: space-separated list of all dependencies
- roscpp - needed to create C++ ROS nodes
- rospy - needed to create Python ROS nodes
- Can add more dependencies later if needed, but roscpp/rospy must be added at creation.
- Generates package.xml, CMakeLists.txt and other folders.

Build the package after creating it.

Robotics Club IITKanpur

Some Common Dependencies

- roscpp, rospy
- Message packages like
  - std_msgs
  - geometry_msgs
  - sensor_msgs
  - nav_msgs
- cv_bridge: allows you to use OpenCV with ROS

Adding Dependencies
(editing *package.xml* and *CMakeLists.txt*)

- Say we need to add a dependency of **std_msgs**.
  We will need to do two things:

```
<depend>std_msgs</depend>
```

1. Edit **package.xml**: add a line like:
   - Preferably at the end of all existing dependencies.

2. Edit **CMakeLists.txt**: in the **find_package()** function, add the name of your dependency.
   - preferably on a new line, after all existing dependencies.

Build the package after doing these edits.

Robotics Club IITKanpur

Adding Dependencies

(editing *package.xml*)

```
51  <buildtool_depend>catkin</buildtool_depend>
52  <build_depend>roscpp</build_depend>
53  <build_export_depend>roscpp</build_export_depend>
54  <exec_depend>roscpp</exec_depend>
55
56
```

Before

```
50  <!--   <doc_depend>doxygen</doc_depend> -->
51  <buildtool_depend>catkin</buildtool_depend>
52  <build_depend>roscpp</build_depend>
53  <build_export_depend>roscpp</build_export_depend>
54  <exec_depend>roscpp</exec_depend>
55
56  <depend>std_msgs</depend>
```

After

Comments will have useful information, don't ignore them

Robotics Club IITKanpur

Adding Dependencies
(editing *CMakeLists.txt*)

```
7   ## Find catkin macros and libraries
8   ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9   ## is used, also find other catkin packages
10  find_package(catkin REQUIRED COMPONENTS
11    roscpp
12  )
```

Before

Instructions are in comments

```
7   ## Find catkin macros and libraries
8   ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9   ## is used, also find other catkin packages
10  find_package(catkin REQUIRED COMPONENTS
11    roscpp
12    std_msgs
13  )
```

After

Build the package to verify your edits.

Robotics Club IITKanpur

Creating Nodes

- Nodes can be either .cpp or .py files
- Reside in the **src** folder of the package
- To a register a node as an executable, we need to edit **CMakeLists.txt** as shown:

```
add_executable([EXECUTABLE_NAME] src/[FILE_NAME].cpp)
target_link_libraries([EXECUTABLE_NAME] ${catkin_LIBRARIES})
```

Read the inline comments to find where to make the edit.

CMake is **case-sensitive. Watch your keystrokes.**

Writing Nodes

- First line is **always:**

```
#include <ros/ros.h>
```

- Followed by **message headers**
  - Suppose you want to use std_msgs/String, need to add:

```
#include <std_msgs/String.h>
```

  - Need to first have the package added as a dependency too.
- Then come the **main** function:

```
int main(int argc, char** argv){
    // Your code here
return 0;}
```

Robotics Club IITKanpur

Writing Nodes - Inside the main function

- First line:
  - Initializes ROS

```
ros::init(argc, argv, "[NODE_NAME]");
```

- Second line:
  - Initializes nodes
  - Can give **namespace** as argument

```
ros::NodeHandle nh;
```

```
ros::NodeHandle nh("~");
Creates node with namespace as /NODE_NAME
```

  - No argument creates node with global namespace

Robotics Club IITKanpur

Writing Nodes - Inside the main function

- Next, create a Rate object
  - Argument specifies rate in Hz

```
ros::Rate loopRate(30);
```

- Finally, a while loop
  - Messages are published
  - Topics are updated
  - Other stuff as well

```
while(ros::ok()){
    // publish message
    // update topics
    loopRate.sleep();
}
```

ros::ok() returns **false** if Ctrl+C is pressed, a shutdown request is received or some other error happens.

Writing Nodes - Publishers

- Publishers write messages onto topics.
- To create a publisher, after you have created a NodeHandle

```
ros::Publisher pub =
nh.advertise<[MESSAGE_TYPE]>("[TOPIC_NAME]", [QUEUE_SIZE]);
```

- Provides a publish function that takes a message object as input

```
pub.publish(msg);
```

  - This goes inside the while loop

Robotics Club IITKanpur

Publisher Example

```
#include <ros/ros.h>
#include <std_msgs/String.h>

int main(int argc, char** argv){
        ros::init(argc, argv, “talker”);
        ros::NodeHandle nh;
        ros::Publisher pub = nh.advertise<std_msgs::String>(“chatter”,10);
        ros::Rate loopRate(30);
        int count = 0;
        while(ros::ok()){
                std_msgs::String msg;
                msg.data = “Hello World ” + std::to_string(count) ;
                pub.publish(msg);
                loopRate.sleep();
                count++;}
return 0;}
```

Robotics Club IITKanpur

Writing Nodes - Subscribers

- Subscribers get data that is published on topics
- To create a subscriber, after you have created a NodeHandle:

```
ros::Subscriber sub = nh.subscribe("[TOPIC_NAME]", [QUEUE_SIZE], [CALLBACK_FUNC_NAME]);
```

  - All subscribers run using a callback function - the last argument
- Inside the while loop:

```
ros::spinOnce();
```

  - Executes all available callback functions

Robotics Club IITKanpur

Writing Nodes - Callback functions

- Created outside the main function: these are functions with void return type
- Takes message object as input - pass by value or by reference

```
void subCallback([MESSAGE_OBJECT] msg){
      // process data here;
return;}
```

- Since callback must be of void type, need to use global variables to store processed data

Robotics Club IITKanpur

Subscriber Example

```
#include <ros/ros.h>
#include <std_msgs/String.h>

std::string data;
void subCallback(std_msgs::String msg){
        data = msg.data;
        ROS_INFO("%s", data.c_str());
return;}

int main(int argc, char** argv){
        ros::init(argc, argv, "listener");
        ros::NodeHandle nh;
        ros::Subscriber sub = nh.subscribe("chatter",
10, subCallback);
        ros::Rate loopRate(30);

        // continued at the right
```

ros::spin(); → Equivalent to
while(ros::ok()) {ros::spinOnce();}
But then can't do any processing outside the callback

ROS_INFO: printf for ROS
data.c_str(): gives C equivalent char array from std::string

```
// continued from left

        while(ros::ok()){
                ros::spinOnce();
                loopRate.sleep();
        }

return 0;}
```

Robotics Club IITKanpur

Contact us if you have any problem/suggestion:

Abhay Varshney	**9559015388**
Madhur Deep Jain	**8894051687**
Neil Shirude	**9850892135**

roboticsclubiitkanpur@gmail.com
http://students.iitk.ac.in/roboclub/
https://www.facebook.com/roboclubiitkanpur
https://www.youtube.com/c/RoboticsClubIITKanpur

Robotics Club IITKanpur