ROS Bootcamp - Day 2

Introduction To ROS1

JC Cruz and Alec Graves

UTSA RAS

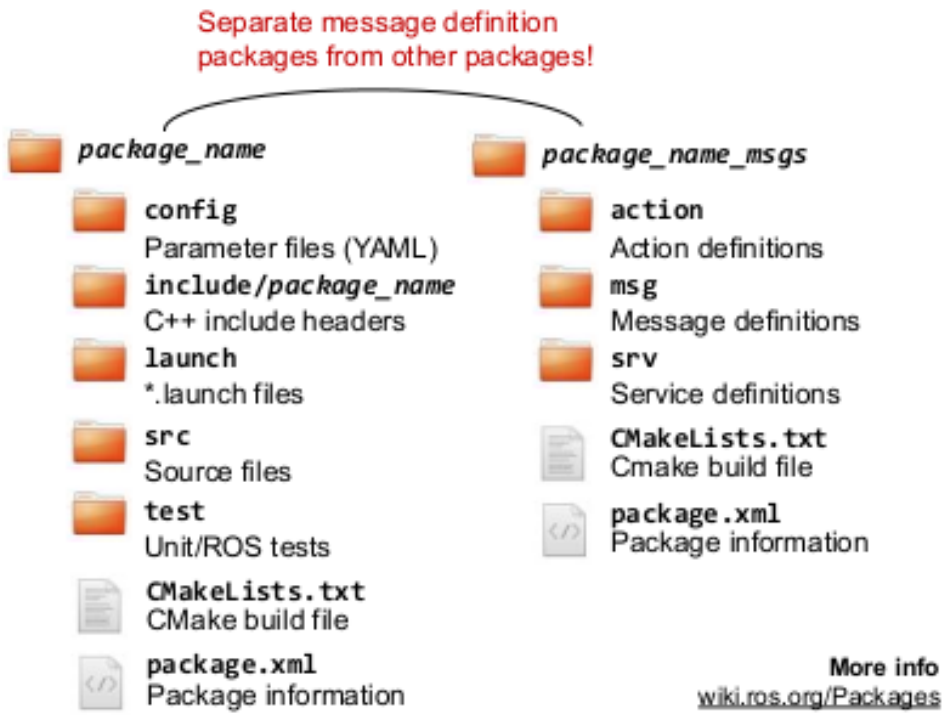
May 16, 2023

Overview

- ROS package structure
- Integration and programming
- ROS subscribers and publishers
- ROS parameter server
- RViz visualization
- **Activity:** IMU Visualization with ESP32

ROS Package Structure

- ROS software organized into packages
- Packages contain: source code, launch files, configuration files, message definitions, data, and documentation
- Packages declare dependencies on other packages
- Use `catkin_create_pkg` to create a new package



Package Directories and Files

- `config`: Parameter files (YAML)
- `include/package_name`: C++ include headers
- `launch`: *.launch files
- `src`: Source files (C++, etc)
- `scripts`: Script file (Python, etc)
- `test`: Unit/ROS tests
- `package_name_msgs`: Message, Service, and Action definitions
- `CMakeLists.txt`: Cmake build file
- `package.xml`: Package information

Package.xml

- `package.xml`: Defines package properties (name, version, authors, dependencies, etc.)

```
<?xml version="1.0"?>
<package format="2">
    <name>ros_package_template </name>
    <version>0.1.0</version>
    <description>A ROS package that... </
     description>
    <maintainer email="youremailt@ethz.ch
     " >You</maintainer>
    <license>BSD</license>
    <url type="website">https://github.
     com/leggedrobotics/ros_...
    </url>

    <buildtool_depend >catkin</
     buildtool_depend >

    <depend>roscpp</depend>
    <depend>std_msgs</depend>

    <build_depend>message_generation </
     build_depend>
</package>
```

CMakeLists.txt

- `CMakeLists.txt`: Input to the CMake build system (required CMake version, package name, configure C++ standard and compile features, find other packages needed for build, message/service/action generators, libraries/executables to build, tests, and install rules)

```
cmake_minimum_required (VERSION 3.10.2 )
project(ros_package_template )

## Use C++14, or 11...
set(CMAKE_CXX_STANDARD 14 )
set(CMAKE_CXX_STANDARD_REQUIRED TRUE )

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

Object Oriented Programming

- OOP is based on "objects" with data (attributes) and code (methods).
- Key OOP principles: encapsulation, inheritance, polymorphism.
- In ROS, nodes often use OOP for structure and modularity.
- ROS allows binding subscribers to class methods:
  ```
  subscriber_ = nodeHandle_.subscribe(topic,
  queue_size, &ClassName::methodName, this);
  ```
- Algorithmic code can be encapsulated in a ROS-independent library for reuse.

ROS C++ Library (roscpp)

- roscpp provides a C++ API for ROS, enabling communication between nodes.
- roscpp handles topics, services, parameters, and transforms.
- To use roscpp, include the header: `#include <ros/ros.h>`
- Declare roscpp as a dependency in CMakeLists.txt and package.xml when building your package.

Initializing and Spinning

- Initialize ROS system with `ros::init(argc, argv, "node_name");`
- Create a node handle to interact with ROS
- Start a loop with `ros::spin()` or `ros::spinOnce()` to process incoming messages
- Use `ros::Rate loop_rate(rate);` to set the loop rate in Hz

```
#include "ros/ros.h"
#include "std_msgs/String.h"

void chatterCallback(const std_msgs::
      String::ConstPtr& msg){
  ROS_INFO("I heard: [%s]", msg->data.
      c_str());
}

int main(int argc, char **argv){
  ros::init(argc, argv, "listener");
  ros::NodeHandle n;

  ros::Subscriber sub = n.subscribe("
      chatter", 1000, chatterCallback);

  ros::spin();
  return 0;
}
```

Node Handles

- Main access point to communicate with the ROS system
- Four main types of node handles:
  1. Default (public) node handle: `nh_ = ros::NodeHandle();`
  2. Private node handle:
     `nh_private_ = ros::NodeHandle("~");`
  3. Namespaced node handle:
     `nh_eth_ = ros::NodeHandle("eth");`
  4. Global node handle:
     `nh_global_ = ros::NodeHandle("/");`

More info: `wiki.ros.org/roscpp/Overview/NodeHandles`

Logging

- Mechanism for logging human-readable text from nodes in the console and to log files
- Instead of `std::cout`, use e.g. `ROS_INFO`
- Automatic logging to console, log file, and `/rosout` topic
- Different severity levels (INFO, WARN, etc.)
- Supports both printf- and stream-style formatting
  - `ROS_INFO("Result: %d", result);` // printf
  - `ROS_INFO_STREAM("Result: " << result);`
- Further features such as conditional, throttled, delayed logging etc.

ROS Subscribers: Overview

- Subscribers receive messages from publishers on a specific topic
- Callback function is called when a message is received, taking the message content as an argument
- Subscribe to a topic using the `subscribe()` method of the node handle
- Keep the subscriber object until you want to unsubscribe
- Use `ros::spin()` to process callbacks; it won't return until the node is shut down

ROS Subscribers: Code Example

```
#include "ros/ros.h"
#include "std_msgs/String.h"

// Callback function to handle incoming messages
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "listener"); // Initialize ROS system
  ros::NodeHandle n; // Main access point to communications with the ROS system

  // Subscribe to the "chatter" topic. Messages are passed to the chatterCallback
     function
  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);

  ros::spin(); // Enter a loop, pumping callbacks

  return 0;
}
```

ROS Publishers: Overview

- Publishers send messages to subscribers on a specific topic
- Create a publisher using the `advertise()` method of the node handle
- Create the message content
- Publish the message using `publisher.publish(message);`

Publisher: Code Example

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

int main(int argc, char **argv){
  ros::init(argc, argv, "talker"); // Initialize ROS system
  ros::NodeHandle n; // Main access point to communications with the ROS system
  // Advertise will be publishing messages on the "chatter" topic to the master
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
  ros::Rate loop_rate(10); // Set loop rate

  int count = 0; // Message count
  while (ros::ok()){
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();

    chatter_pub.publish(msg); // Publish the message

    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

ROS Parameter Server

- Nodes use the parameter server to store and retrieve parameters at runtime
- Best used for static data such as configuration parameters
- Parameters can be defined in launch files or separate YAML files
- List all parameters with `> rosparam list`
- Get the value of a parameter with
  `> rosparam get parameter_name`
- Set the value of a parameter with
  `> rosparam set parameter_name value`

More info: `wiki.ros.org/rosparam`

Parameters

- Get a parameter in C++ with `nodeHandle.getParam(parameter_name, variable)`
- Method returns true if parameter was found, false otherwise
- Global and relative parameter access
- For parameters, typically use the private node handle `ros::NodeHandle("~")`

More info: `wiki.ros.org/roscpp/Overview/Parameter`

RViz

- 3D visualization tool for ROS
- Subscribes to topics and visualizes the message contents
- Different camera views (orthographic, top-down, etc.)
- Interactive tools to publish user information
- Save and load setup as RViz configuration
- Extensible with plugins

Run RViz with `> rviz`

Activity

Find Today's activity sheet at: \docs\lec2

`https://github.com/UTSARobotics/ros1_bootcamp`