CSC752 Autonomous Robotic Systems
- Introduction into ROS (2) -

Ubbo Visser

Department of Computer Science
College of Arts and Sciences
University of Miami

September 2022
OVERVIEW

- ROS package structure
- ROS C++ client library (roscpp)
- ROS subscribers and publishers
- ROS parameter server
- RViz visualization
ROS software is organized into packages, which can contain source code, launch files, configuration files, message definitions, data, and documentation.

A package that builds upon or requires other packages (e.g. message definitions), declares these as dependencies.

Creating a package:

```
~$catkin_create_pkg package_name {dependencies}
```
The `package.xml` file hosts and defines the properties of a package:

- Name of the package
- Versioning
- Authors
- Dependencies
- ...

```xml
<?xml version="1.0"?>
<package format="2">
  <name>hsrb_uv</name>
  <version>0.0.1</version>
  <description>The HSRB-UV package</description>
  <maintainer email="visser@cs.miami.edu">Ubbo Visser</maintainer>
  <license>BSD</license>
  <build>catkin</build>
  <build>roscpp</build>
  <build>sensor_msgs</build>
  <build>roscpp</build>
  <build>sensor_msgs</build>
  <exec>roscpp</exec>
  <exec>sensor_msgs</exec>
</package>
```

Details at: [http://wiki.ros.org/catkin/package.xml](http://wiki.ros.org/catkin/package.xml)
ROS PACKAGES

- The CMakeLists.txt file defines the necessary inputs for the build system
  - Required CMake Version
  - Package Name
  - Find other CMake/Catkin packages needed for build
  - Message/Service/Action Generators (add_message_files(), add_service_files(), add_action_files())
  - Invoke message/service/action generation (generate_messages())
  - Specify package build info export
  - Libraries/Executables to build (add_library()/add_executable()/target_link_libraries())
  - Tests to build (catkin_add_gtest())
  - Install rules (install())

Details at:
http://wiki.ros.org/catkin/CMakeLists.txt

cmake_minimum_required(VERSION 3.0.2)
project(hsr_b_uv)

add_compile_options(-std=c++11)

find_package(catkin REQUIRED COMPONENTS
  roscpp
  sensor_msgs)

include_directories(
  include
  ${catkin_INCLUDE_DIRS}
)

add_executable(${PROJECT_NAME}
  src/hsrb_uid_node.cpp
  src/HSRBController.cpp)

target_link_libraries(${PROJECT_NAME}
  ${catkin_LIBRARIES})
ROS C++ CLIENT LIBRARY (ROSCPP)

- ROS main header file include
- `ros::init(...)` has to be called before calling other ROS functions.
- The node handle is the access point for communications with the ROS system (topics, services, parameters)
- `ros::Rate` is a helper class to run loops at a desired frequency
- `ros::ok()` checks if a node should continue running
  Returns false if SIGINT is received (Ctrl + C) or `ros::shutdown()` has been called
- ROS_INFO() logs messages to the filesystem
- `ros::spinOnce()` processes incoming messages via callbacks

Details at:
There are four main types of node handles:

- Default (public) node handle:
  \[\text{nh\_} = \text{ros::NodeHandle();}\]

- Private node handle:
  \[\text{nh\_private\_} = \text{ros::NodeHandle("~");}\]

- Namespaced node handle:
  \[\text{nh\_rc\_} = \text{ros::NodeHandle("rc");}\]

- Global node handle:
  \[\text{nh\_global\_} = \text{ros::NodeHandle("/");}\]

For a node in namespace looking up topic, these will resolve to:

- `/namespace/topic`
- `/namespace/node/topic`
- `/namespace/rc/topic`
- `/topic`

Not recommended

Details at:
http://wiki.ros.org/roscpp/Overview/NodeHandles
ROS C++ CLIENT LIBRARY (ROSCPP) 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, Error etc.)
- Supports both printf- and stream-style formatting

```cpp
ROS_INFO("Result: %d", result);
ROS_INFO_STREAM("Result: " << result);
```

- Further features such as conditional, throttled, delayed logging etc.

<table>
<thead>
<tr>
<th></th>
<th>Debug</th>
<th>Info</th>
<th>Warn</th>
<th>Error</th>
<th>Fatal</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>std::cout</code></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>stderr</code></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Log file</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><code>/rosout</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

To see output in terminal: use launch file to configure:

```xml
<launch>
  <node name="listener" output="screen"/>
</launch>
```

Details at:
http://wiki.ros.org/roscpp/Overview/Logging
Start listening to a topic by calling the method subscribe() of the node handle

```cpp
ros::Subscriber subscriber = nodeHandle.subscribe(topic, queue_size, callback_function);
```

When a message is received, the callback function is called with the contents of the message as the argument

Hold on to the subscriber object until you want to unsubscribe

`ros::spin()` processes callbacks and will not return until the node has been shutdown

Details at:
http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers
- Create a publisher with help of the node handle

```cpp
ros::Publisher publisher =
  nodeHandle.advertise<message_type>(topic,
queue_size);
```

- Create the message contents

- Publish content with

```cpp
publisher.publish(message)
```

Details at:
http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers
Specify a function handler to a method from within the class as

```cpp
subscriber = nodeHandle.subscribe(topic, queue_size, &ClassName::methodName, this);
```

Details at:
http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers
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 parameters with**

```bash
~$rosparam list
```

**Get values**

```bash
~$rosparam get parameter_name
```

**Set values**

```bash
~$rosparam set parameter_name
```

### config.yaml

```yaml
camera:
  left:
    name: left_camera
    exposure: 1
  right:
    name: right_camera
    exposure: 1.1
```

### package.launch

```xml
<launch>
  <node name="name" pkg="package" type="node_type"/>
  <rosparam command="load" file="$(find package)/config/config.yaml"/>
</launch>
```

Details at:

[http://wiki.ros.org/rosparam](http://wiki.ros.org/rosparam)
Get a parameter in C++ with

```cpp
nodeHandle.getParam(parameter_name, variable)
```

Method returns `true` if parameter was found, `false` otherwise.

Global and relative parameter access:

- Global parameter name with preceding `/`

  ```cpp
  nodeHandle.getParam("/package/camera/left/exposure", variable)
  ```

- Relative parameter name (relative to the node handle)

  ```cpp
  nodeHandle.getParam("camera/left/exposure", variable)
  ```

- For parameters, typically use the private node handle
  `ros::NodeHandle("~")`

Details at:
http://wiki.ros.org/roscpp/Overview/Parameter%20Server
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:

~$rosrun rviz rviz

Details at: http://wiki.ros.org/rviz
FURTHER REFERENCES

- ROS Wiki
  - http://wiki.ros.org/
- Installation
  - http://wiki.ros.org/ROS/Installation
- Tutorials
  - http://wiki.ros.org/ROS/Tutorials
- Packages
  - https://www.ros.org/browse/list.php
- ROS Cheat Sheet
  - https://kapeli.com/cheat_sheets/ROS.docset/Contents/Resources/Documents/index
- ROS Best Practices
  - https://github.com/leggedrobotics/ros_best_practices/wiki
- ROS Package Templates
  - https://github.com/leggedrobotics/ros_best_practices/tree/master/ROS_package_template
Material is based on ROS Wiki and ETH Zürich ROS Introduction (https://rsl.ethz.ch/)