CSC398 Autonomous Robots
- Introduction into ROS (2) -

Ubbo Visser

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

September 2024

UNIVERSITY OF MIAMI
ROBOCANES

OVERVIEW

- ROS package structure
- ROS C++ client library (roscpp)
- ROS subscribers and publishers
- ROS parameter server
- RViz visualization

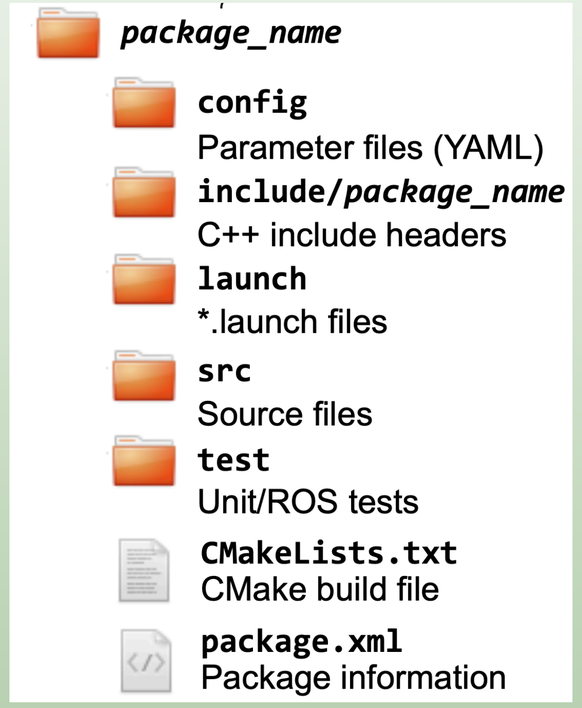
ROS PACKAGES

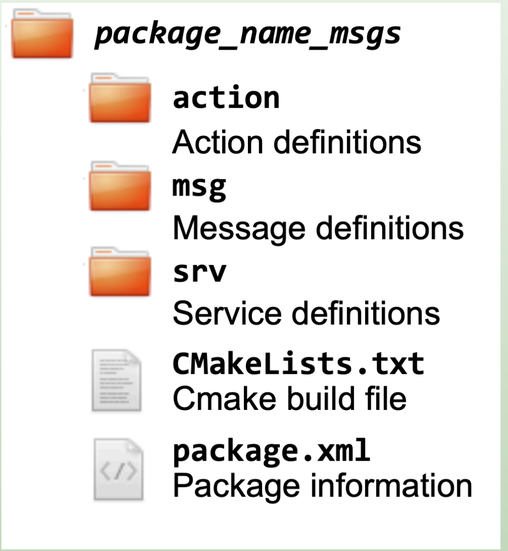
- ROS software is organized into packages, which can contain source code, launch files, configuration files, message definitions, data, and documentation
- A package that builds up on or requires other packages (e.g. message definitions), declares these as dependencies

Creating a package:

```
~$catkin_create_pkg package_name {dependencies}
```

Separate message definition packages from other packages!





Details at: http://wiki.ros.org/Packages

ROS PACKAGES

- 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 HSB package</description>
  <maintainer email="visser@cs.miami.edu">Ubbo Visser</maintainer>
  <license>BSD</license>
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>rospy</build_depend>
  <build_depend>sensor_msgs</build_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>sensor_msgs</build_export_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>sensor_msgs</exec_depend>
</package>
```

```
In [CATKIN_WS]/src:
catkin_create_pkg hsrb_uv rospy sensor_msgs
```

Details at:
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(hsrb_uv)

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

## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
catkin_python_setup()

###################################
## catkin specific configuration ##
###################################
## Declare things to be passed to dependent projects
## CATKIN_DEPENDS: catkin packages dependent projects also need
catkin_package(
  CATKIN_DEPENDS rospy sensor_msgs std_msgs
)

###########
## Build ##
###########

## Specify additional locations of header files
include_directories(
  ${catkin_INCLUDE_DIRS}
)

#############
## Install ##
#############

## Mark executable scripts (Python etc.) for installation
catkin_install_python(PROGRAMS
  src/publisher1.py
  src/minscan.py
  src/minscanrviz.py
  src/minscanparams.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

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(hsrb_uv)

add_compile_options(-std=c++11)

find_package(catkin REQUIRED COMPONENTS
  roscpp
  sensor_msgs
)

################################
## catkin specific configuration ##
################################

catkin_package(
  INCLUDE_DIRS
    include
  CATKIN_DEPENDS
    roscpp
    sensor_msgs
)

###########
## Build ##
###########

include_directories(
  include
  ${catkin_INCLUDE_DIRS}
)

add_executable(${PROJECT_NAME}
 src/hsrb_uv_node.cpp
 src/HSRBUVController.cpp
)

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

ROS C++ CLIENT LIBRARY (ROSCPP)

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

int main(int argc, char** argv)
{
  ros::init(argc, argv, "hello_world");
  ros::NodeHandle nodeHandle;
  ros::Rate loopRate(10);

  unsigned int count = 0;
  while (ros::ok())
  {
    ROS_INFO_STREAM("Hello World " << count);
    ros::spinOnce();
    loopRate.sleep();
    count++;
  }
  return 0;
}
```

- 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:
http://wiki.ros.org/roscpp, http://wiki.ros.org/roscpp/Overview

ROS C++ CLIENT LIBRARY (ROSCPP) NODE HANDLE

- There are four main types of node handles
  - Default (public) node handle:
    nh_ = ros::NodeHandle();
  - Private node handle:
    nh_private_ = ros::NodeHandle("~");
  - Namespaced node handle:
    nh_rc_ = ros::NodeHandle(“rc");
  - Global node handle:
    nh_global_ = ros::NodeHandle("/");

Not recommended

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

  `/namespace/topic`

  `/namespace/node/topic`

  `/namespace/rc/topic`

  `/topic`

Details at:
http://wiki.ros.org/roscpp/Overview/NodeHandles

ROS PYTHON CLIENT LIBRARY (ROSPY) 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

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

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

	Debug	Info	Warn	Error	Fatal
stdout	X	X			
stderr			X	X	X
Log file	X	X	X	X	X
/rosout	X	X	X	X	X

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

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

Details at:
http://wiki.ros.org/rosconsole, http://wiki.ros.org/roscpp/Overview/Logging

ROS C++ CLIENT LIBRARY (ROSCPP) SUBSCRIBER

- Start listening to a topic by calling the method subscribe() of the node handle

```
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

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

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

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

  ros::Subscriber subscriber =
            nodeHandle.subscribe("chatter",10,chatterCallback);
  ros::spin();

  return 0;
}
```

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

ROS C++ CLIENT LIBRARY (ROSCPP) PUBLISHER

- Create a publisher with help of the node handle

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

- Create the message contents
- Publish content with

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

```
#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 chatterPublisher =
        nh.advertise<std_msgs::String>("chatter", 1);
  ros::Rate loopRate(10);

  unsigned int count = 0;

  while (ros::ok())
  {
    std_msgs::String message;
    message.data = "hello world " + std::to_string(count);
    ROS_INFO_STREAM(message.data);
    chatterPublisher.publish(message);
    ros::spinOnce();
    loopRate.sleep();
    count++;
  }
  return 0;
}
```

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

ROS C++ CLIENT LIBRARY (ROSCPP) OOP

```cpp
#include <ros/ros.h>
#include "my_package/MyPackage.hpp"

int main(int argc, char** argv)
{
  ros::init(argc, argv, "my_package");
  ros::NodeHandle nodeHandle("~");

  my_package::MyPackage myPackage(nodeHandle);

  ros::spin();

  return 0;
}
```

MyPackage.hpp

MyPackage.cpp

`class MyPackage`

Main node class providing ROS interface (subscribers, parameters, timers etc.)

Algorithm.hpp

Algorithm.cpp

`class Algorithm`

Class implementing the algorithmic part of the node

Note: The algorithmic part of the code could be separated in a (ROS-independent) library

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 PYTHON CLIENT LIBRARY (ROSPY)

```python
import rospy
from std_msgs.msg import String
import random


def getName():
    names = ['John', 'Doe', 'Jane', 'Smith', 'Alice']
    return random.choice(names)

def getStatus():
    status = ['happy', 'sad', 'angry', 'excited', 'bored', 'tired', 'hungry',
              'awake']
    return random.choice(status)

rospy.init_node('publisher_node')
publisher = rospy.Publisher('/ubbo1', String, queue_size=1)

def callback(event):
    msg = String()
    msg.data = '%s is %s' %(getName(), getStatus())
    publisher.publish(msg)
    rospy.loginfo(msg.data)

rospy.Timer(rospy.Duration(1), callback)
rospy.spin()
```

- ROS import rospy
- **rospy.init_node(…)** has to be called before calling other ROS functions.
- r**osy.publisher(…)** is publishing the topic
- r**osy.spin ()** checks if a node should continue running. Returns false if SIGINT is received (Ctrl + C) or rospy.shutdown() has been called
- ROS_INFO() logs messages to the filesystem
- r**ospy.Timer()** processes incoming messages via callbacks

Details at:
http://wiki.ros.org/rospy

ROSPY CLIENT LIBRARY (ROSY) LOGGING

- Mechanism for logging human readable text from nodes in the console and to log files
- Instead of **std::cout**, use e.g. rospy.loginfo()
- Automatic logging to console, log file, and **/rosout topic**
- Different severity levels (Info, Warn, Error etc.)

```
rospy.loginfo(msg.data)
```

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

	Debug	Info	Warn	Error	Fatal
stdout	X	X			
stderr			X	X	X
Log file	X	X	X	X	X
/rosout	X	X	X	X	X

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

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

Details at:
http://wiki.ros.org/rosconsole, http://wiki.ros.org/roscpp/Overview/Logging

ROSPY CLIENT LIBRARY SUBSCRIBER

- Start listening to a topic by calling the method subscribe() of the node handle

```
rospy.Subscriber('/hsrb/base_scan', LaserScan, callback)
```

- 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
- r**ospy.spin()** processes callbacks and will not return until the node has been shutdown

```
import rospy
from sensor_msgs.msg import LaserScan

rospy.init_node('laser_scan_listener')
rospy.Subscriber('/hsrb/base_scan', LaserScan, callback)

def callback(scan):
    # Ensure the range data is not empty and find the minimum range value
    if scan.ranges:
        min_distance = TODO
        rospy.loginfo("Minimum distance to object: %.2f meters", min_distance)
    else:
        rospy.logwarn("No range data available.")

rospy.Timer(rospy.Duration(30), callback)
rospy.spin()
```

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

ROSPY CLIENT LIBRARY PUBLISHER

- Create a publisher with help of the node handle

```
publisher = rospy.Publisher('/ubbo1', String, queue_size=1)
```

- Create the message contents
- Publish content with

```
publisher.publish(msg)
```

```
import rospy
try:
    from std_msgs.msg import String # type: ignore
except ImportError:
    rospy.logerr("Failed to import 'std_msgs.msg'")
import random


def getName():
    names = ['John', 'Doe', 'Jane', 'Smith', 'Alice']
    return random.choice(names)

def getStatus():
    status = ['happy', 'sad', 'angry', 'excited', 'bored']
    return random.choice(status)

rospy.init_node('publisher_node')
publisher = rospy.Publisher('/ubbo1', String, queue_size=1)

def callback(event):
    msg = String()
    msg.data = '%s is %s' %(getName(), getStatus())
    publisher.publish(msg)
    rospy.loginfo(msg.data)

rospy.Timer(rospy.Duration(1), callback)
rospy.spin()
```

Details at:
http://wiki.ros.org/rospy/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

```
~$rosparam list
```

Get values

```
~$rosparam get parameter_name
```

Set values

```
~$rosparam set parameter_name
```

config.yaml

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

package.launch

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

Details at:
http://wiki.ros.org/rosparam

ROS PARAMETER SERVER C++ API

- 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:
  - Global parameter name with preceding /

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

  - Relative parameter name (relative to the node handle)

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

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

```
ros::NodeHandle nodeHandle("~");
std::string topic;

if (!nodeHandle.getParam("topic", topic))
{
  ROS_ERROR("Could not find topic parameter!");
}
```

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

- Get a parameter in C++ with

```
rospy.get_param(parameter_name, variable)
```

- Method returns *true* if parameter was found, *false* otherwise
- Global and relative parameter access:
  - Global parameter name with preceding /

```
rospy.get_param('/package/camera/left/exposure', variable)
```

  - Relative parameter name (relative to the node handle)

```
rospy.get_param('camera/left/exposure', variable)
```

- Set parameters

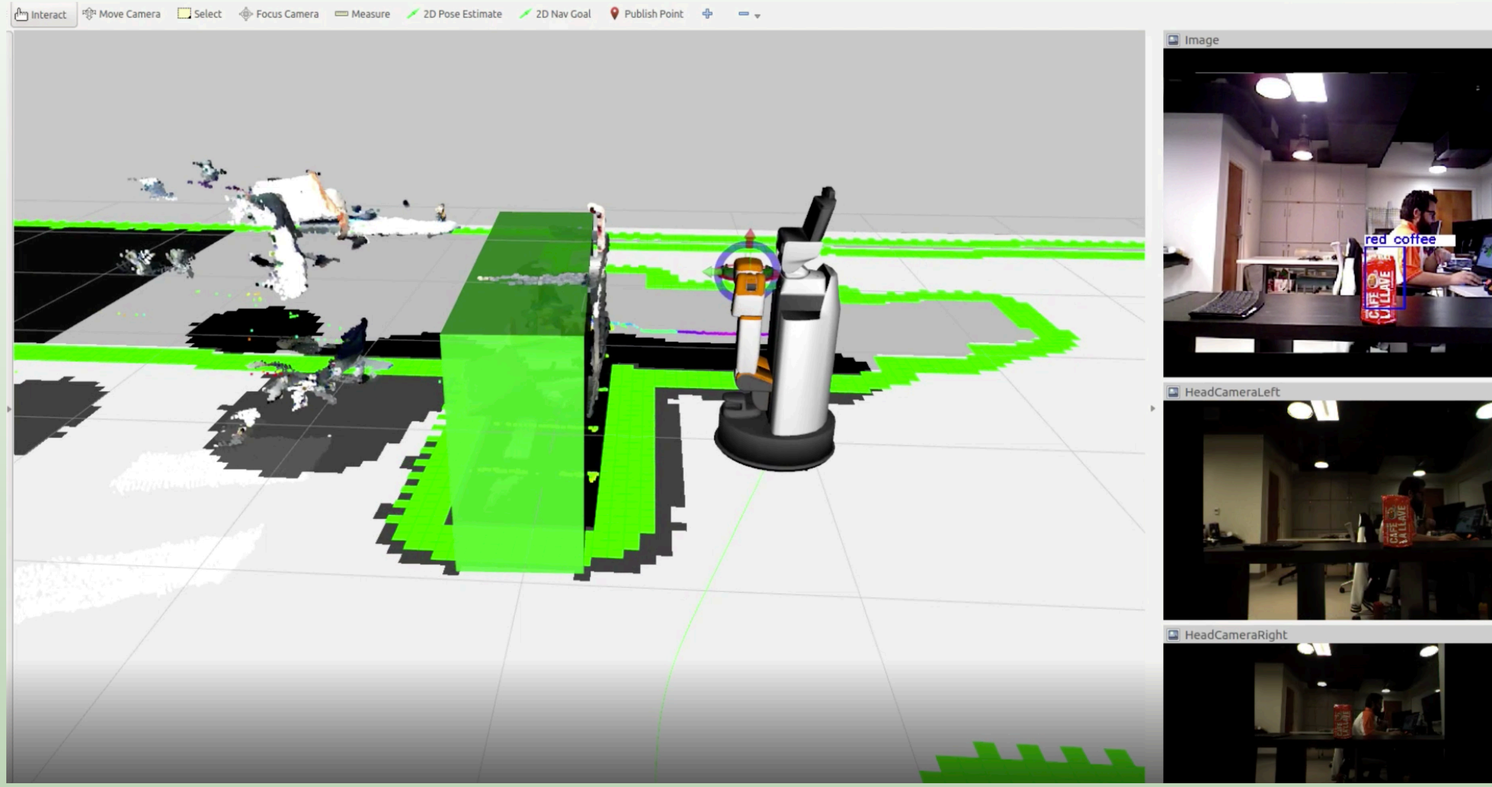
```
rospy.set_param('bool_True', True)
```

```
try:
    rospy.get_param_names()
except ROSException:
    print("could not get param name")
```

Details at:
http://wiki.ros.org/rospy/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://www.clearpathrobotics.com/ros-robot-operating-system-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

ACKNOWLEDGEMENT

Material is based on ROS Wiki and ETH Zürich ROS Introduction (https://rsl.ethz.ch/)